CR 114486

AVAILABLE TO THE PUBLIC

HYBRID CODING SYSTEMS STUDY

FINAL REPORT

(NASA-CR-114486) HYBRID CODING SYSTEMS STUDY Final Report J.P. Odenwalder, et al (Linkabit Corp.) Sep. 1972 139 p CSCL 09B

N72-32206

Unclas

G3/08 42039

LINKABIT CORPORATION

10453 ROSELLE STREET
UNIVERSITY INDUSTRIAL PARK
SAN DIEGO, CALIFORNIA 92121

TABLE OF CONTENTS

Section		Page No.
1.0	INTRODUCTION	1
2.0	CONCATENATED CODING AND DECODING	8
	2.1 Operation	9
	2.2 Performance	11
	2.3 Implementation	25
	2.3.1 Encoder and Interleaver Design	27
	2.3.2 Unscrambler and Decoder Storage Implementation	29
	2.3.3 Reed-Solomon Decoding Procedure	32
	2.4 Part Software and Part Hardware Decoder Implementation	36
	2.5 Hardware Implementation	40
	2.5.1 Hardware Implementation of Field Operations	40
	2.5.2 Hardware Encoder and Interleaver Design	45
	2.5.3 Synchronization Implementation	49
	2.5.4 Hardware Unscrambler Implementation	50
	2.5.5 Hardware Reed-Solomon Decoder Design	50
	2.5.5.1 Syndrome Calculation	53
	2.5.5.2 Berlekamp Algorithm Implementation	53
	2.5.5.3 Chien Search and Error Evaluation Preparation	61
	2.5.5.4 Chien Search and Error Evaluation	63
	2.5.6 Hardware Implementation Summary	66
3.0	HYBRID BOOTSTRAP DECODING	68
	3.1 Performance Results	68
	2 Hybrid Bootstrap Sequential Decoder Implementation	74
	3.2.1 Decoder Memory Organization	76
	3.2.2 Decoding Logic	77
	3.2.3 Track Control Logic	84
	3.2.4 Parts Count Estimation	87
	3.2.5 Decoder Computation Rate	88
	3.3 Other Bootstrap Decoding Techniques	89
	3.3.1 Multiple Processors	89
	3.3.2 Bootstrap Trellis Decoding	90
	3.3.2.1 Description of Rudimentary Decoder	90
	3.3.2.2 Analytical Performance Estimates	92
	3.3.2.3 Refinements of the Decoding Algorithm	96
4.0	CONCLUSIONS AND RECOMMENDATIONS	103
	APPENDIX A	106
	REFERENCES	134

TABLE OF FIGURES

Figure No.	Title	Page No.
2.1.1	Concatenated Coding System	10
2.2.1	Concatenated Coding Performance with a K=7, R=1/2 Inner Code and 6 bits/R-S symbol	16
2.2.2	Concatenated Coding Performance with a K=7, R=1/2 Inner Code and 7 bits/R-S symbol	17
2.2.3	Concatenated Coding Performance with a K=7, R=1/2 Inner Code and 8 bits/R-S symbol	18
2.2.4	Concatenated Coding Performance with a K=7, R=1/2 Inner Code and 9 bits/R-S symbol	19
2.2.5	Summary of Concatenated Coding Performance with K=7, R=1/2 Inner Code	21
2.2.6	Summary of Concatenated Coding Performance with K=8, R=1/2 Inner Code and a 2^J-Symbol E-Error Correcting Outer Code	22
2.2.7	Summary of Concatenated Coding Performance with K=8, R=1/3 Inner Code and a 2^J-Symbol E-Error Correcting Outer Code	23
2.2.8	Summary of Concatenated Coding Performance with K=8, R=1/7 Inner Code and a 2^J-Symbol E-Error Correcting Outer Code	24
2.3.1	Outer Code Array	26
2.3.2	Outer Encoder and Interleaver	28
2.3.3	Unscrambler Implementation	30
2.3.4	Procedure for Calculating the i^{th} Syndrome	34
2.5.1	Alternate Field Multiplication Procedure	43
2.5.2	Alternate Field Multiplication Implementation	44
2.5.3	Outer Encoder and Interleaver Implementation	46
2.5.4	Reed-Solomon Decoder Block Diagram	51
2.5.5	Reed-Solomon Decoder Timing Diagram	52
2.5.6	Syndrome Calculation	54
2.5.7	Berlekamp Algorithm Block Diagram	56
2.5.8	d_n Calculation Procedure	57
2.5.9	Block Diagram of the Main Processor of the Berlekamp Algorithm	58
2.5.10	Berlekamp Algorithm Implementation	60
2.5.11	Chien Search and Error Evaluation Preparation	62
2.5.12	Chien Search and Error Evaluation Procedure	64
3.1.1	Extrapolated Distributions, Octal Hybrid Sequential Decoder	70
3.1.2	Erasure Probability for 15 MCPS Hybrid Sequential Decoder, Rate 1/3, Octal Channel	73
3.2.1	Hybrid Bootstrap Sequential Decoder	75
3.2.2	Hybrid Bootstrap Sequential Decoder Branch	78
3.2.3	Complementary Distribution Function for MT for rate 1/2, inner code, 7-track Hybrid Bootstrap Decoder	79
3.2.4	Sensitivity of Bootstrap Decoder Computations to Quantization of Metrics and of KLEFT	80

TABLE OF TABLES

Table No.	Title	Page No.
1.0.1	Summary of Decoder Performance	2
2.4.1	Software R-S Decoder Speeds	38
2.5.1	Summary of the estimated number of chips required to hardware implement a concatenated coding system with a K=8, R=1/3 convolutional inner code and a J=8, E=16 R-S outer code with I=16 levels of interleaving	67
3.1.1	Approximate IC Requirements for Hybrid Bootstrap Sequential Decoding	87
A.1	LINKABIT Supplement to the IBM 1130 Assembly Language Instruction Set	108

1.0 Introduction

With the growth of digital space communication in the past decade, the introduction of sophisticated coding techniques has provided efficiency improvements which have resulted in reductions of required power or extended communication range for numerous space missions. While early coding applications were for relatively low data rates, recent emphasis has been on real-time decoders capable of operation at data rates above 1 Mbps and even approaching 100 Mbps. These efforts have resulted in the development of high speed decoders which provide on the order of 4 to 6 dB of coding gain depending on the data rate, code rate or bandwidth expansion, and error probability requirements.

The left half of Table 1.0.1 summarizes the present state of efficiency improvement available with high speed decoders presently in operation or under development. The required ratio of bit-energy-to-noise-density, E_b/N_o, is given in each case for bit error probabilities of 10^{-4} and 10^{-7}*.

When the data speed requirements are reduced to the levels of deep space applications, which are on the order of from 1 Kbps to 100 Kbps, greater coding gains can be achieved. At these reduced speeds, sequential decoding particularly can be shown to operate more efficiently.

*Only convolutional codes are considered here. Block codes which were common in early applications are so definitely inferior both in required complexity and in resulting performance that their further treatment is not worthwhile for the systems under consideration, other than as outer codes in a concatenated coding system.

DECODER	UNCODED	SEQUENTIAL (HARD DECISION)	VITERBI	VITERBI	SEQUENTIAL (SOFT DECISION)	CONCATENATED VITERBI	HYBRID BOOTSTRAP SEQUENTIAL	
Quantization/levels	2	2	8	8	8	8	8	
Convolutional Code Rate	-	1/2	1/2	1/3	1/3	1/3	1/3	
Constraint length for V.D.	-	-	7	8	-	8	-	
Buffer Size for S.D. and concatenated	-	64 K	-	-	64 K	64 K*	64 K*	
Sequential decoder computation rate in Mega computations/sec.	-	100	-	-	1	-	15	
Principal Logic Family	-	MECL III +MECL 10,000 +TTL	TTL	TTL	TTL	TTL	MECL 10,000	
Data Rate	-	40 Mbps	2 to 8 Mbps	100 Kbps	100 Kbps	100 Kbps†	10 Kbps	100 Kbps
E_b/N_o (dB) for $P_B = 10^{-4}$	8.4	5.0	3.7	3.0	2.8	1.93	1.7	2.2
E_b/N_o (dB) for $P_B = 10^{-7}$	11.3	5.7	5.6	4.8	3.4	2.18	2.0	3.2
Approximate Number of IC's	-	700	250	150	400	333	450	

*Half of this storage is for the current block being received, while the other half is for the previous block being decoded.

†At speeds of 10 Kbps or below there is a potential improvement of up to 0.2 dB, but only if the storage buffer is essentially doubled in size.

Table 1.0.1 Summary of Convolutional Decoder Performance

A hard quantized high speed sequential decoder can be operated with about 1 dB less E_b/N_o, because of the increased number of computations per bit period. Further performance can be gained at lower speeds by using soft (8 or more level) quantization and thus regaining most of the 2 dB loss inherent in hard (2 level) quantization. Also, more efficient Viterbi decoders are possible at reduced data rates, although the improvement in this case is not as great.

The potential performance of low rate decoders is shown in the middle columns of Table 1.0.1. For the sequential decoder, we consider a code-rate 1/3 system. Assuming a computation speed of 1 Megacomputations/second on soft decision data, a 64 K bit buffer, and a 500 bit block length with frame resynchronization, we find an improvement of about 2.3 dB relative to the hard decision, code-rate 1/2, high speed sequential decoder operating at 40 Mbps. The improvement is about 1 dB less if both are operating at 100 Kbps. For the Viterbi decoder, we consider a constraint-length 8, code-rate 1/3 decoder which is considerably less complex than the low rate sequential decoder. Its performance is equivalent or better for bit error rates above 10^{-4}, but it becomes progressively worse at low error rates. Improvements in either system through increased complexity (larger buffer and higher computation speed with ECL logic for the sequential decoder - higher constraint lengths with greatly increased path and metric memory requirements for the Viterbi decoder) are very costly and could gain on the order of 0.5 dB.

A more promising approach at low data rates is the use of concatenated or hybrid coding and decoding techniques. This study deals with the performance and implementation of two particularly promising techniqes, shown in the right-hand part of Table 1.0.1. Each is based essentially on one of the decoders just discussed, augmented by an additional device (block decoder for the concatenated system - control logic and additional metric calculators for the hybrid system) whose complexity is not greater than that of the original decoders. Yet the resulting improvement is much greater than would be possible if the original decoders were simply upgraded by increasing the complexity or speed in the manners discussed above.

Some of the conclusions are summarized in the two rightmost columns of Table 1.0.1. The performance of the two systems are remarkably similar and the required buffer sizes are approximately the same. The concatenated approach appears to require about one third fewer IC's, and these are of the TTL rather than of the MSI ECL logic family. The latter are required by the hybrid system because of the high required speed factor of the sequential decoder. These advantages are partially offset by the fact that the concatenated system requires several read-only memory (ROM) and random-access memory (RAM) chips which are relatively expensive.

Otherwise, it actually appears that the concatenated system is preferable and that it is even cost-competitive with a simple sequential decoder, while achieving approximately a 1 dB performance gain on the latter. All the systems in the three rightmost columns require approximately the same buffer size. In only two respects the concatenated system may be inferior to the other two: namely, while the sequential decoder generally requires about a 30 bit synchronization sequence (tail) for approximately every 500 data bits, and the hybrid bootstrap decoder requires about a 164 bit synchronization sequence for approximately every 3000 bits, the concatenated decoder in the preferred form requires a 4096 bit non-data sequence (consisting primarily of outer code parity checks) every 28,672 data bits. These long gaps in the data stream may not be significantly disturbing when many users are time-division multiplexed together, but may represent a serious drawback when only one data stream is sent. This problem can almost certainly be alleviated by using a staggered interleaving scheme. Unfortunately, this requires the simultaneous (though still serial) decoding of several outer code words. A secondary and corollary effect is that the decoding delay in the concatenated system is of the order of 32 to 64 Kbits, while for the sequential and hybrid sequential systems, it is only on the order of the 64K of buffer storage which corresponds only to about 7000 bits.

Finally, it should be noted that an ideal rate 1/3 eight-level soft decision coded system operating at channel capacity requires a bit energy-to-noise density of -0.3 dB. This means that the two systems under consideration are operating at about 2.5 dB from the ultimate capacity (or Shannon limit) of the coding format. Thus, it appears from Table 1.0 that at $P_b = 10^{-7}$, there are almost 12 dB of ultimate coding gain between the uncoded system and the ideal coded system operating at channel capacity. With the first level of sophistication (leftmost third) involving coding with rate 1/2 codes, which may operate up to multi-megabit data rates, almost half this gain is achievable. With the second level of sophistication (middle third) involving code rate 1/3 lower data rates, longer codes for Viterbi decoding, and soft rather than hard decision sequential decoding, an additional 1 to 2 dB are gained. Beyond this, the third level of sophistication under consideration here gains another 1.5 to 2 dB at $P_b = 10^{-7}$. Thus, obviously another such step-function increase is just not possible. Experience in this study and previously has convinced us of the futility and frustration in further attempts in reducing the small gap left in achievable coding gain. The next "breakthrough," if it ever occurs, might be worth another 0.5 dB. As will be discussed in Section 4.0, we conclude that, on the basis of present theory and technology, the concatenated or hybrid coding systems under consideration can be realized in a cost-effective manner and are certain to stand as the ultimate in coding gain for space communication systems far into the foreseeable future.

This final report is organized as follows. In Section 2 we treat concatenated coding and decoding, beginning with a review of the principles of operation and a detailed analysis of performance with various configurations. We then consider several possible implementations and concentrate on a detailed evaluation of the preferred hardware implementation. In Section 3, we proceed in the same way for hybrid coding and decoding. Section 4 presents our conclusions and recommendations.

2.0 Concatenated Coding and Decoding

The principle of concatenated coding and decoding as a means of reducing the number of errors in received data in two or more successive stages began with Elias' iterative coding procedures (Ref. 1). They were extended for block codes by numerous researchers, the most complete study being that of Forney (Ref. 2). Pinsker first (Ref. 3) and later Stiglitz (Ref. 4) considered concatenation of convolutional and block codes, using a block code as the inner (first stage) code in an attempt to improve the channel, so as to increase the computational cutoff rate R_{comp} for the sequential decoder operating on the outer (second stage) code. While this produced interesting theoretical results, it requires a very complex and impractical inner decoder. A much more reasonable approach is to use the more efficient and powerful code - the convolutional code - internally and thus, for a given complexity, improve the channel as much as possible for the outer decoder. While the outer decoder may also be convolutional, the resulting "super channel" consisting of the original channel with inner coder and decoder seems especially well suited to a particular class of block codes over a multiple alphabet discovered by Reed and Solomon (Ref. 5). This technique used with Viterbi decoding was investigated by Odenwalder (Ref. 6) and found to yield rather impressive results. In the remainder of this section, we concentrate on this approach

2.1 Operation

The basic block diagram is shown in Figure 2.1.1. The inner coder-decoder is a short constraint length convolutional coder with a Viterbi (maximum likelihood) decoder. Typically this decoder is operated at an E_b/N_o level sufficient to produce a bit error probability in the range $10^{-2}>P_b>10^{-3}$. The outer code is a high rate (low redundancy) block code which then reduces the final block, and consequently bit, error probability to the desired level. The most efficient class of codes found for this purpose are the Reed-Solomon (R-S) codes with a block length of 2^J-1 symbols over a 2^J-ary alaphabet, where the best choice of J appears to be approximately equal to the constraint length of the inner coder. The interleaving buffers are required because the inner decoder errors tend to occur in bursts, which occasionally are as long as several constraint lengths. While the outer decoder is undisturbed by burst errors within a given 2^J-ary symbol (which corresponds to J bits or about one constraint length), its performance is severely degraded by highly correlated errors among several successive symbols; hence the need for interleaving.

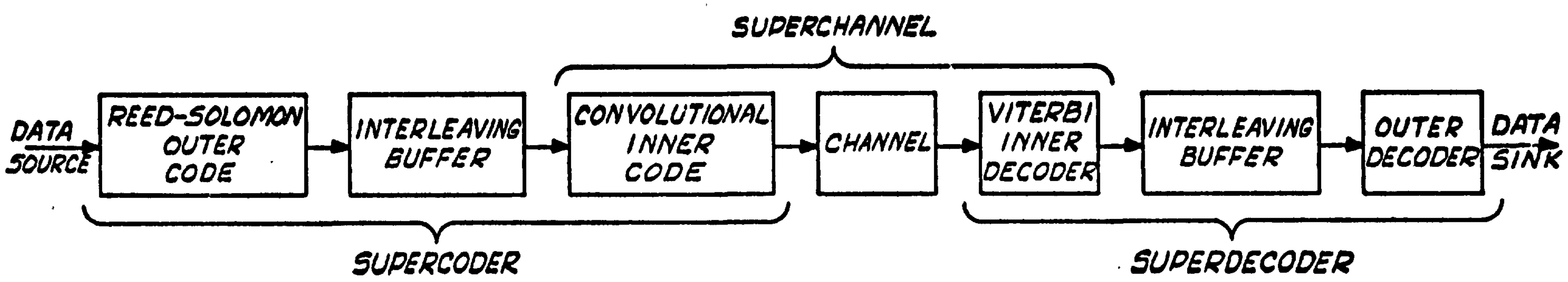


Figure 2.1.1 Concatenated Coding System.

2.2 Performance

To evaluate the performance of this concatenated coding system under cost and complexity constraints, the significant parameters of the inner code are the R-S symbol error probability and the distribution of lengths of consecutive R-S symbol errors, the latter being required to determine the required interleaver dimensions.

Both experimentally and theoretically a more directly derived indication of inner code performance is the distribution of error lengths in bits. The length of an error-burst for a convolutional code of constraint length K is naturally defined as the number of bits starting with the initial error and terminating when K-1 consecutive correct bits have been received. Let this distribution of bit error burst lengths be denoted by

$$Q_k = \Pr \text{ (at any node an error-burst of length } k \text{ terminates)} \tag{2.2.1}$$

We desire to determine the distribution of lengths of consecutive R-S symbol errors, P_j, from the bit error burst distribution Q_k.

To determine P_j we must recognize, first of all, that the error bursts on the inner convolutional code are totally asynchronous to the outer code symbol phase. Suppose then that the first incorrect R-S symbol begins m bits prior to the start of the convolutional code bit rror burst. Because of the asynchronous nature of the situation, m is a uniformly distributed random variable on the interval $0 \leq m \leq J - 1$. Now conditioning on a fixed m we have

$$P_j(m) = \sum_{k=(j-1)J-m+1}^{jJ-m} Q_k \qquad , j = 1, 2, \ldots \quad (2.2.2)$$

and we define

$$Q_k = 0 \text{ for } k \leq 0$$

Summing on the variable m, we have therefore,

$$P_j = \sum_{m=0}^{J-1} P_j(m) = \sum_{m=0}^{J-1} \sum_{k=(j-1)J-m+1}^{jJ-m} Q_k \quad , j=1,2 \ldots \qquad (2.2.3)$$

If $J > K-1$, this expression is exact since every subsequence of J bits must contain at least one error and hence cause the R-S symbol to be in error. On the other hand, if $J < K-1$, some R-S symbol in the sequence may possibly be correct, so that (2.2.3) becomes an over estimate at the high end of the distribution.

To obtain the R-S symbol error probability from the error length distribution, we need to weigh P_j by the number of errors in each case. Since, as pointed out above, we take all consecutive symbols to be in error, we have for the symbol error probability

$$\Pi_s = \sum_{j=1}^{\infty} jP_j \tag{2.2.4}$$

Also from (2.2.3) we can obtain the desired interleaving length. For example, if we require that the ultimate (outer code) error probability be P_b, then we should take the interleaver length L in R-S symbols to be such that

$$P_L < P_b \tag{2.2.5}$$

Finally, assuming a long enough interleaver so that we can neglect error dependencies, the output bit error probability can be bounded as follows. For an E-error-correcting R-S outer code, a R-S block error occurs when more than E symbol errors occur in the block. When this happens, the R-S decoder indicates that at most E symbols are in error. So, if the superchannel causes E+i, $1 \leq i \leq 2^J - 1 - E$, symbol errors in the block, at most 2E + i symbol errors will result. Thus, the concatenated code symbol probability of error can be upper bounded by

$$P_s \leq \sum_{i=E+1}^{2^J-1} (i+E)\binom{2^J-1}{i} \Pi_s^i \; (1-\Pi_s)^{2^J-1-i} \tag{2.2.6}$$

Since some of the bits in an incorrect symbol may be correct, the bit probability of error is less than or equal to the symbol probability of error. The symbol errors caused by the R-S decoder will have about half their bits in error, while those caused by the superchannel will typically have from .25 to .40 of their bits in error, depending on the particular inner code and channel. Here we will simply upper bound the bit probability of error by the symbol probability of error. Thus,

$$P_b \leq \sum_{i=E+1}^{2^J-1} (i+E)\binom{2^J-1}{i} \Pi_s^i \quad (1-\Pi_s)^{2^J-1-i} \qquad (2.2.7)$$

To cover the data rates of interest and to provide the performance data needed to optimize this system for various complexity constraints, the following inner codes were simulated.

1) K=7, R=1/2 with code generators
1 1 1 1 0 0 1
1 0 1 1 0 1 1

2) K=8, R=1/2 with code generators
1 1 1 1 1 0 0 1
1 0 1 0 0 1 1 1

3) K=8, R=1/3 with code generators
1 1 1 1 0 1 1 1
1 1 0 1 1 0 0 1
1 0 0 1 0 1 0 1

4) K=8, R=1/7 with code generators
1 1 1 1 1 0 0 1
1 0 1 0 0 1 1 1
1 1 1 1 0 1 1 1
1 1 0 1 1 0 0 1
1 0 0 1 0 1 0 1
1 0 0 1 1 1 1 1
1 1 1 0 0 1 0 1

The code generators in Cases 1, 2, and 3 are those obtained by Odenwalder in Reference 6. These code generators were chosen to minimize the bit probability of error at large E_b/N_o ratios. However, in the range of E_b/N_o's used here other codes could yield better results.

The code generators in Case 4 were obtained using the code generators of Odenwalder's rate 1/3 code, his rate 1/2 code, and the reciprocals of his rate 1/2 code. In this case, this yields a code with a free distance of 38, which is close to the upper bound of 40 which Heller (Reference 7) has obtained on the free distance of K=8, R=1/7 codes.

These simulations were for convolutional coding systems with practically implementable Viterbi decoders (Reference 8) using 8 levels of receiver quantization and a path length memory of 32 bits. The bit error burst length statistics were computed and equations 2.2.3 through 2.2.7 were used to compute the R-S symbol probability of error, the distribution of lengths of consecutive R-S symbol errors, and the bit probability of error bound.

Figures 2.2.1 through 2.2.4 give the concatenated code bit probability of error bound for a K=7, R=1/2 convolutional code and 6, 7, 8, and 9 bits per R-S symbol, respectively. They show that for a fixed alphabet size and probability of error, there is an optimum number

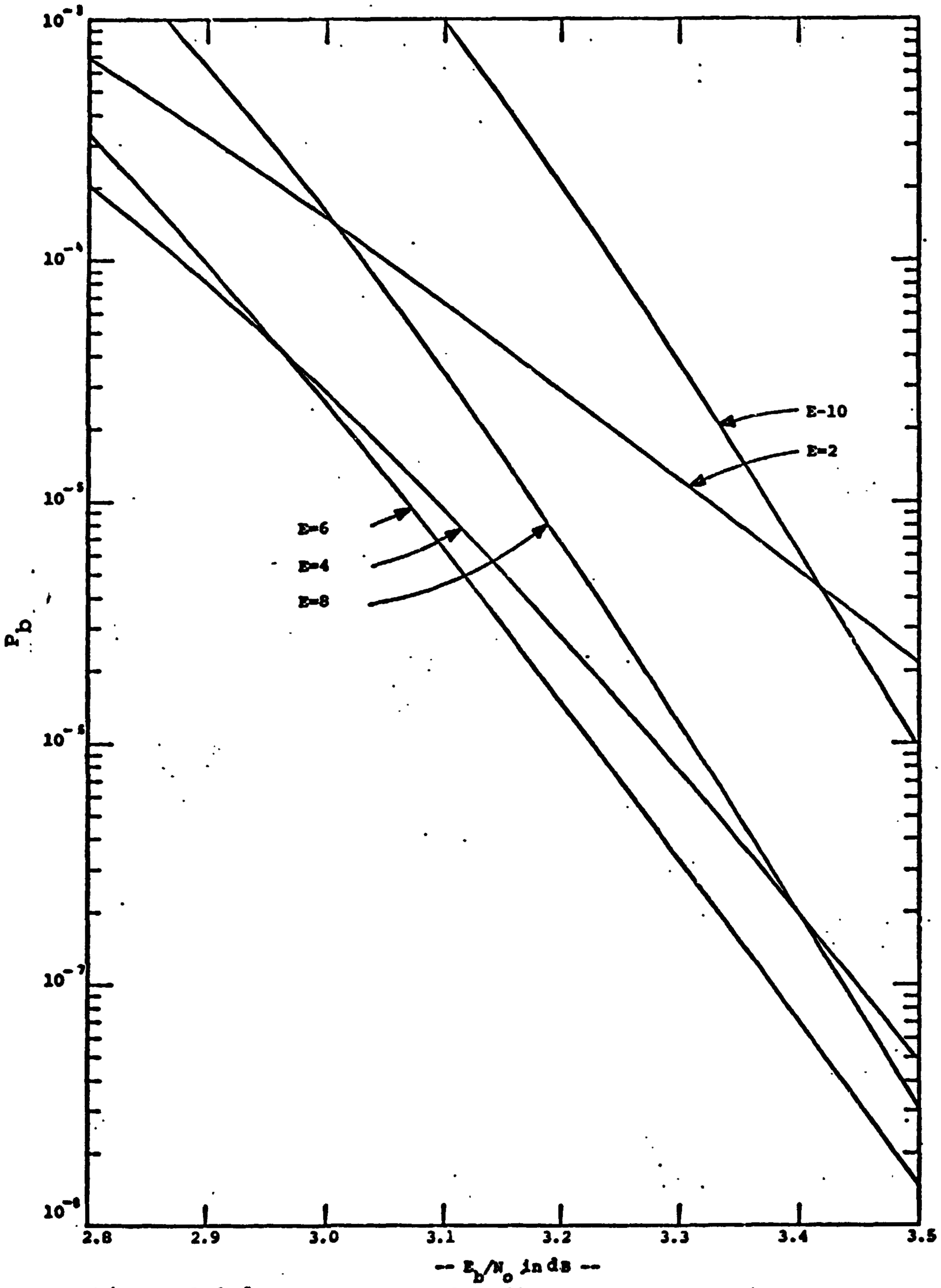


Figure 2.2.1. Concatenated Coding Performance with a K=7, R=1/2 Inner Code and 6 Bits/R-S Symbol.

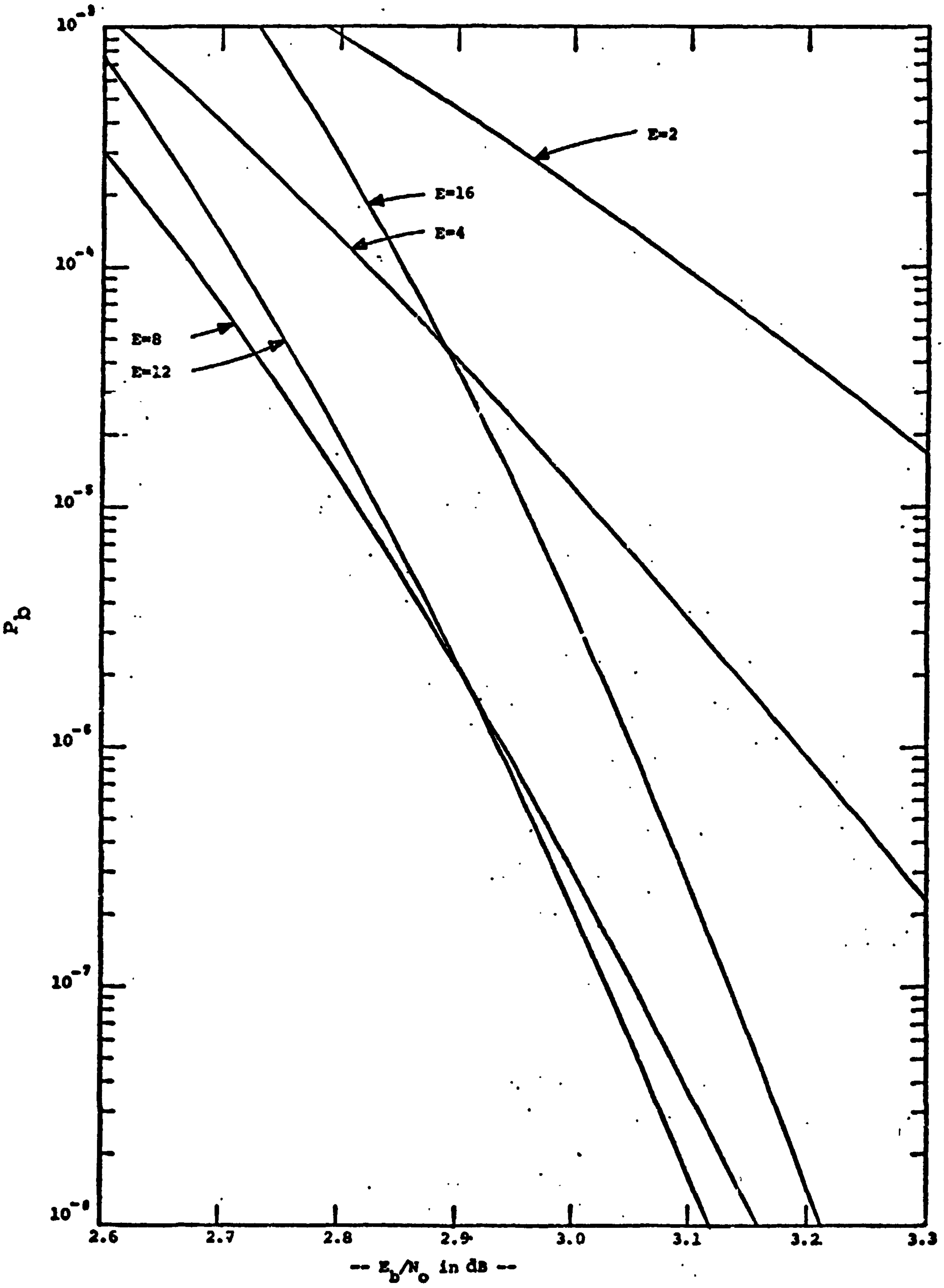


Figure 2.2.2. Concatenated Coding Performance with a K=7, R=1/2 Inner Code and 7 Bits/R-S Symbol.

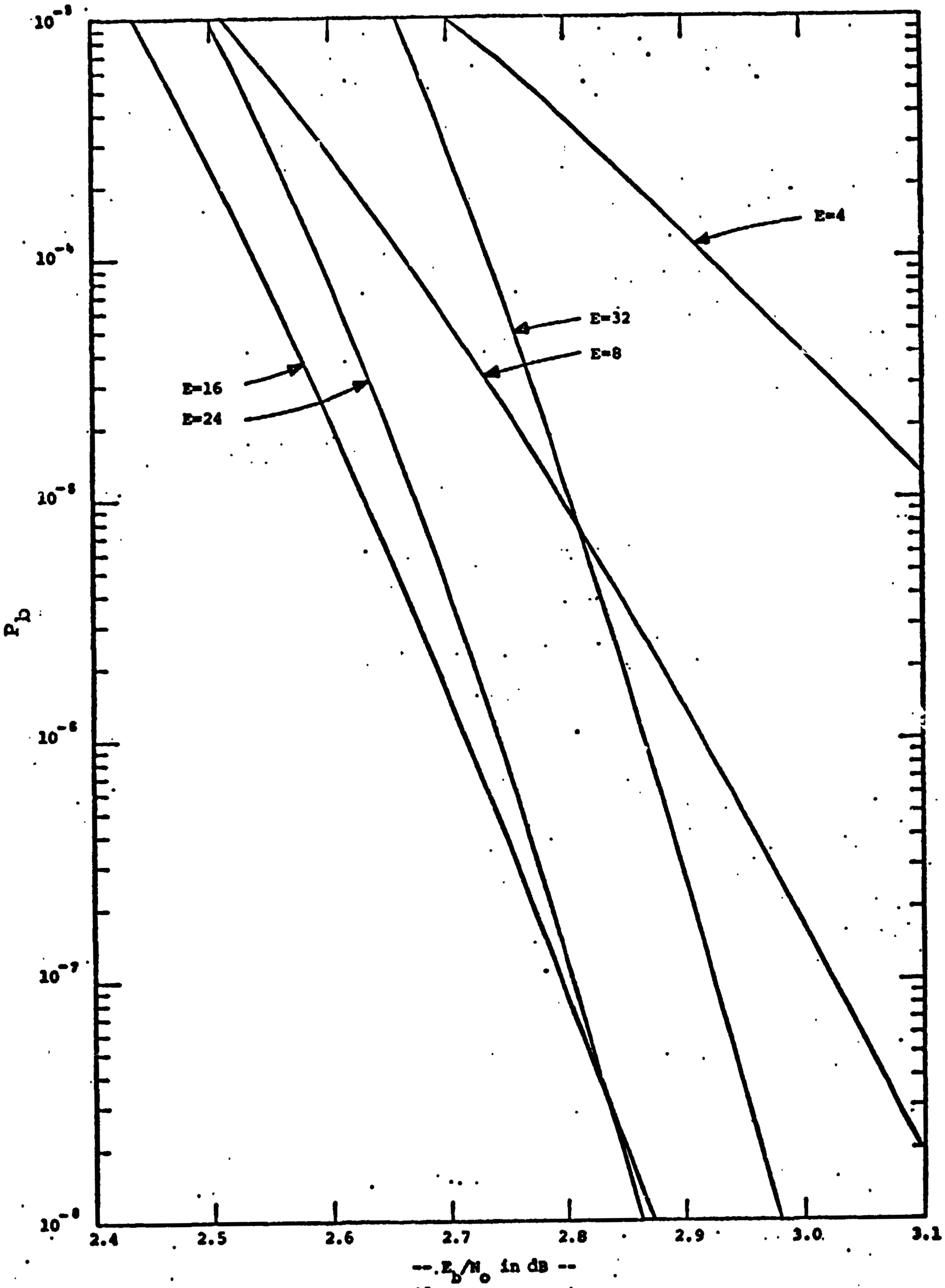


Figure 2.2.3. Concatenated Coding Performance with a K=7, R=1/2 Inner Code and 8 Bits/R-S Symbol.

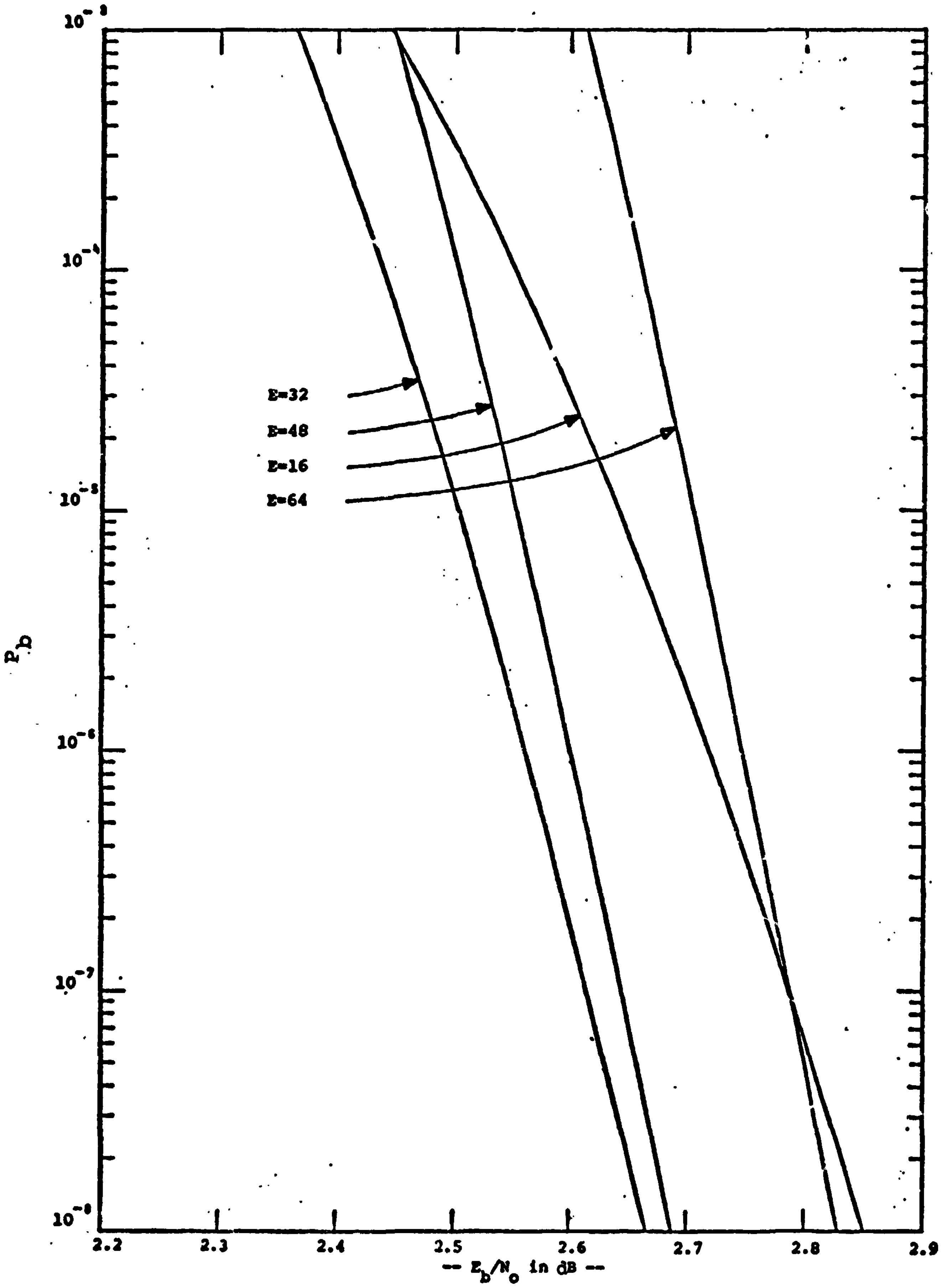


Figure 2.2.4. Concatenated Coding Performance with a K=7, R=1/2 Inner Code and 9 Bits/R-S Symbol.

of correctable errors. That is, if the outer code is designed to correct too many errors, the inner code E_b/N_o decrease, and thus the superchannel symbol probability of error increase, more than offsets the large error correcting ability of the outer code. These curves also show in some cases that it may be desirable to design the outer decoder to correct less than the optimum number of correctable errors. Such a system would require a larger E_b/N_o ratio to achieve a specified probability of error, but the decoder would be faster and easier to implement.

Figures 2.2.5 through 2.2.8 summarize the performance of this concatenated coding system for the four convolutional inner codes, various alphabet sizes, and near optimum outer code error correcting ability.

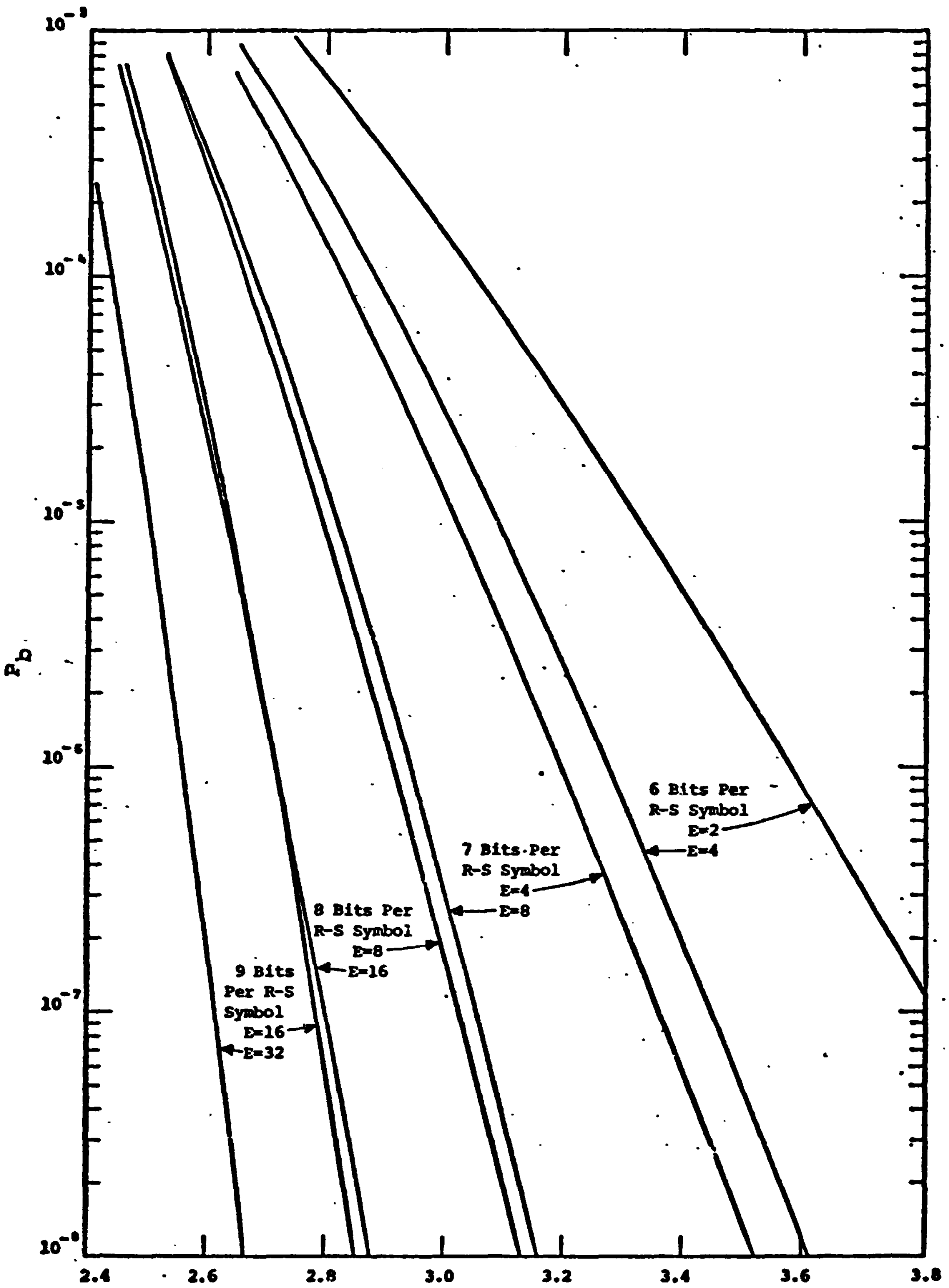


Figure 2.2.5. Summary of Concatenated Coding Performance with a K=7, R=1/2 Inner Code.

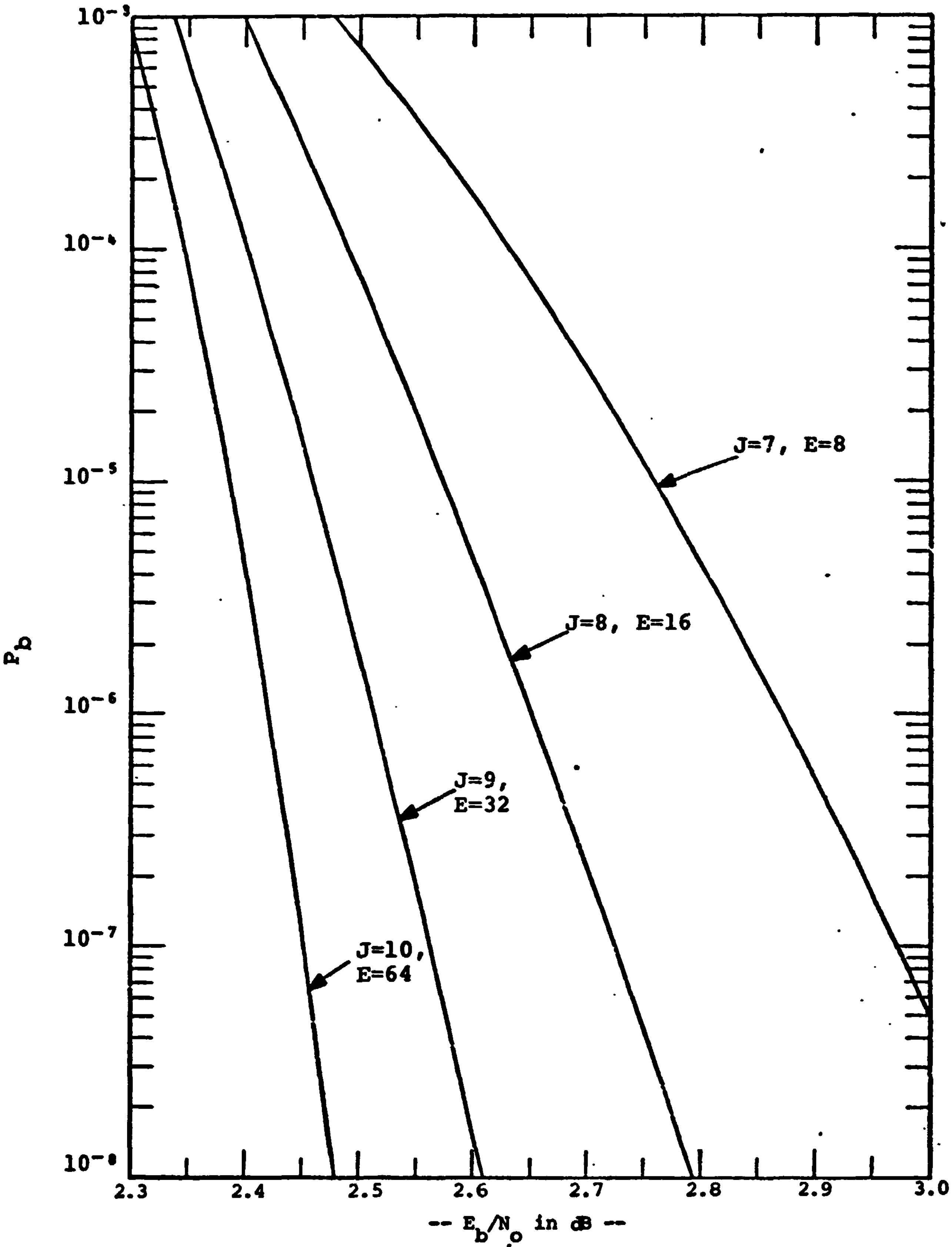


Figure 2.2.6. Summary of Concatenated Coding Performance with a K=8, R=1/2 Inner Code and a 2^J-Symbol E-Error-Correcting Outer Code.

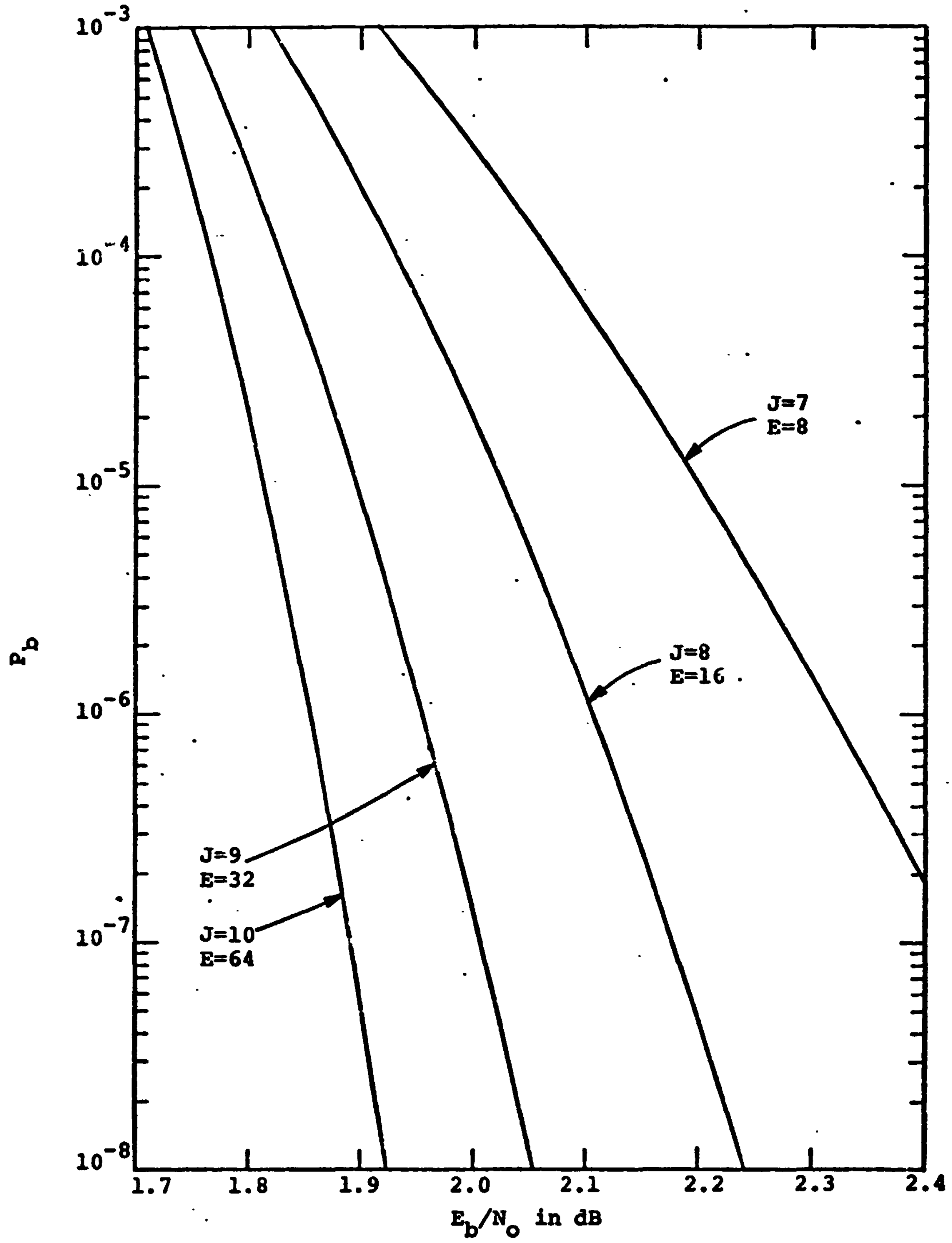


Figure 2.2.7. Summary of Concatenated Coding Performance with a K=8, R=1/3 Inner Code and a 2^J-Symbol E-Error-Correcting Outer Code.

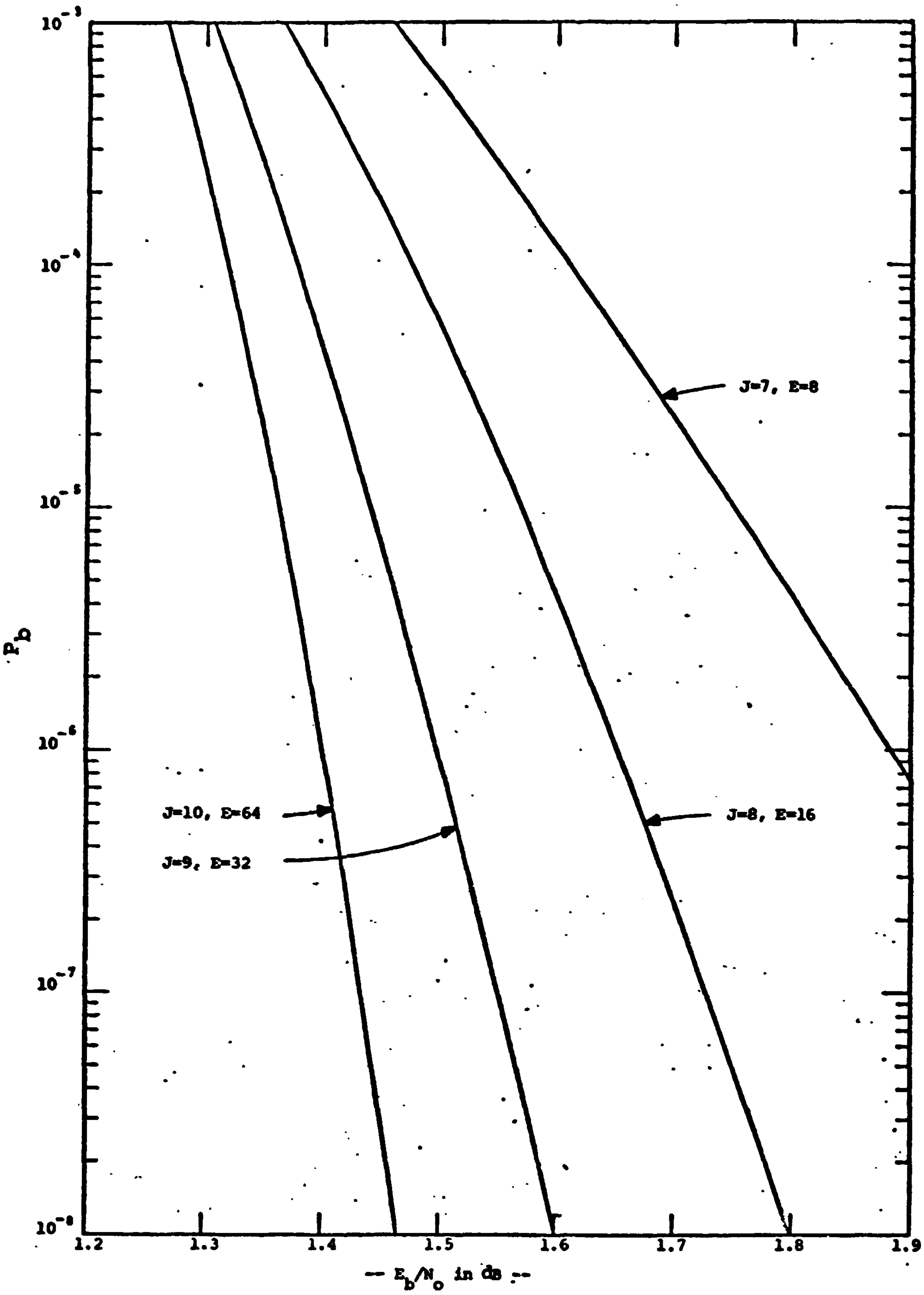


Figure 2.2.8. Summary of Concatenated Coding Performance with a K=8, R=1/7 Inner Code and a 2^J-Symbol E-Error-Correcting Outer Code.

2.3 Implementation Procedure

At data rates up to 100 Kbps, hardware implementation of constraint length 7 and 8 Viterbi decoders is relatively straightforward. LINKABIT has implemented a K=7, R=1/2 Viterbi decoder* using only 85 IC's and the implementation of K=8 Viterbi decoders is documented in References 8 and 9. So most of the system design here is concerned with the outer coder-decoder and the interleaving buffers. For the present purpose, the inner coder-decoder can be regarded as part of the channel. The inner code rate and constraint length have virtually no effect on the outer code design, except to the small extent that longer constraint lengths cause longer error bursts and hence require longer interleaving.

The basic outer code parameters are summarized in the code structure diagram of Figure 2.3.1. Each of the I rows in this array represents a R-S code word of 2^J-1, J-bit symbols followed by a J-bit segment of a synchronization sequence. This assumes, of course, that the data to be transmitted can be interrupted periodically for the insertion of the (2E+1)JI parity and synchronization bits. This will be the case, for example, when several users are time-division multiplexed together. I is the degree of interleaving, chosen sufficiently long to ensure the independence of successive horizontal R-S symbols, E is the guaranteed number of correctable R-S symbol errors, and 2E is the required

* It is estimated that a K=8, R=1/3 Viterbi decoder can be implemented for data rates up to 100 Kbps with 150 TTL IC's.

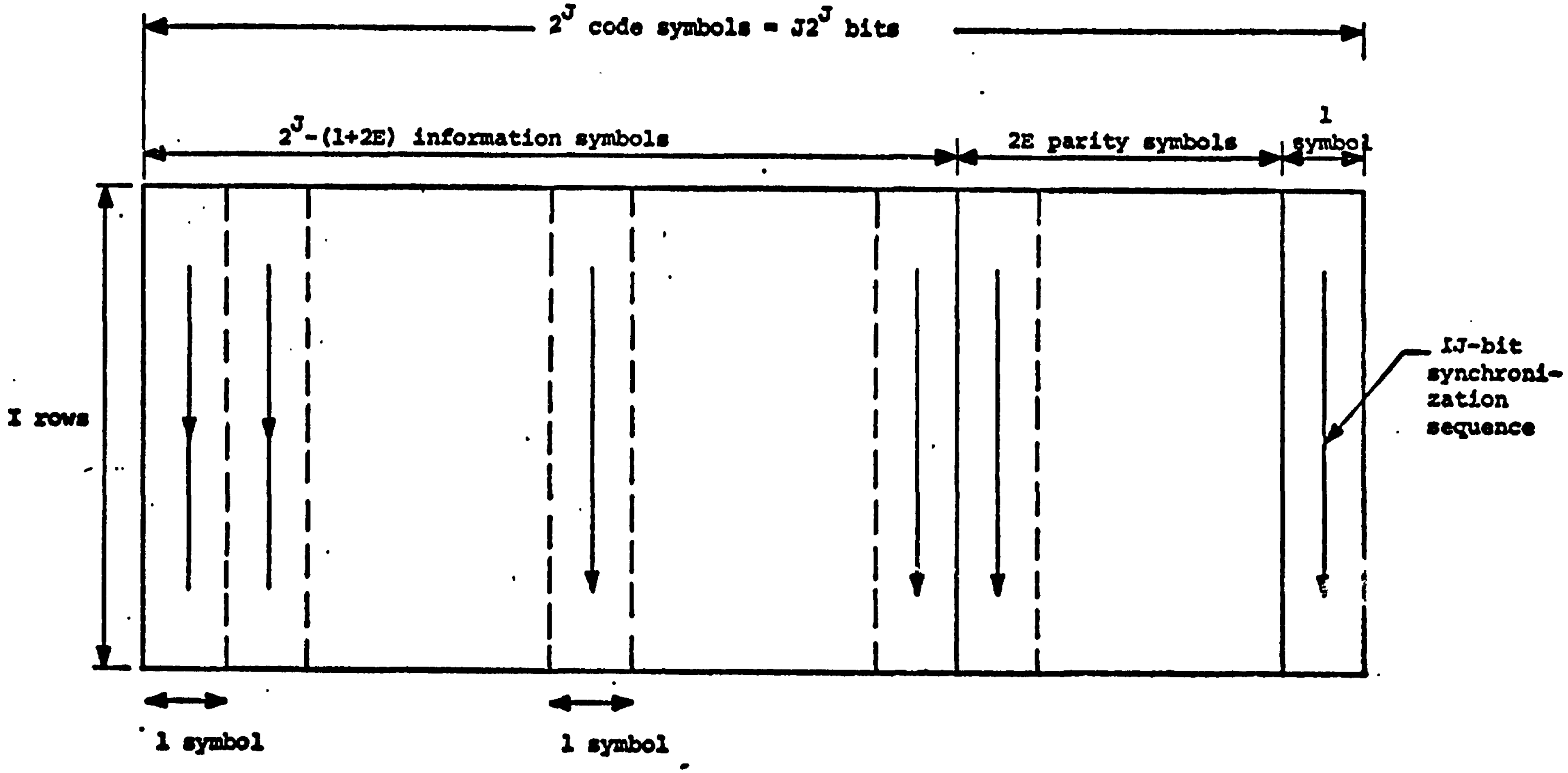


Figure 2.3.1 Outer Code Array.

number of parity checks. It is assumed that the data is presented to the encoder in blocks of $IJ(2^J-1-2E)$ information bits followed by a period where the 2EIJ parity bits and the IJ-bit synchronization sequence can be inserted. The encoded bits are read out of the array in blocks of J bits (one R-S symbol) one column at a time and fed to the inner convolutional encoder.

Code synchronization is obtained using the IJ-bit synchronization sequence of Figure 2.3.1 and the synchronization ability of the Viterbi decoder. The Viterbi decoder provides inner code node synchronization and phase ambiguity resolution as described in Reference 8. Then the IJ-bit sequence of superchannel symbols is used to obtain code array, and thus R-S symbol, synchronization. However, due to the bursty nature of the superchannel, several code arrays may have to be examined to obtain the code array synchronization.

2.3.1 Encoder and Interleaver Design

The encoding and interleaving operations can be accomplished as shown symbolically in Figure 2.3.2. This basic encoder is the most efficient for a cyclic code with 2E parity checks when $2E<2^J-1-2E$ (see Figure 6.5.5 of Reference 10). The double lines represent J-bit signal flow and the additions and multiplications are over $GF(2^J)$. The generator polynomial is

$$g(D) = g_o+g_1 D+ \dots +g_{2E-1}D^{2E-1} \qquad (2.3.1)$$

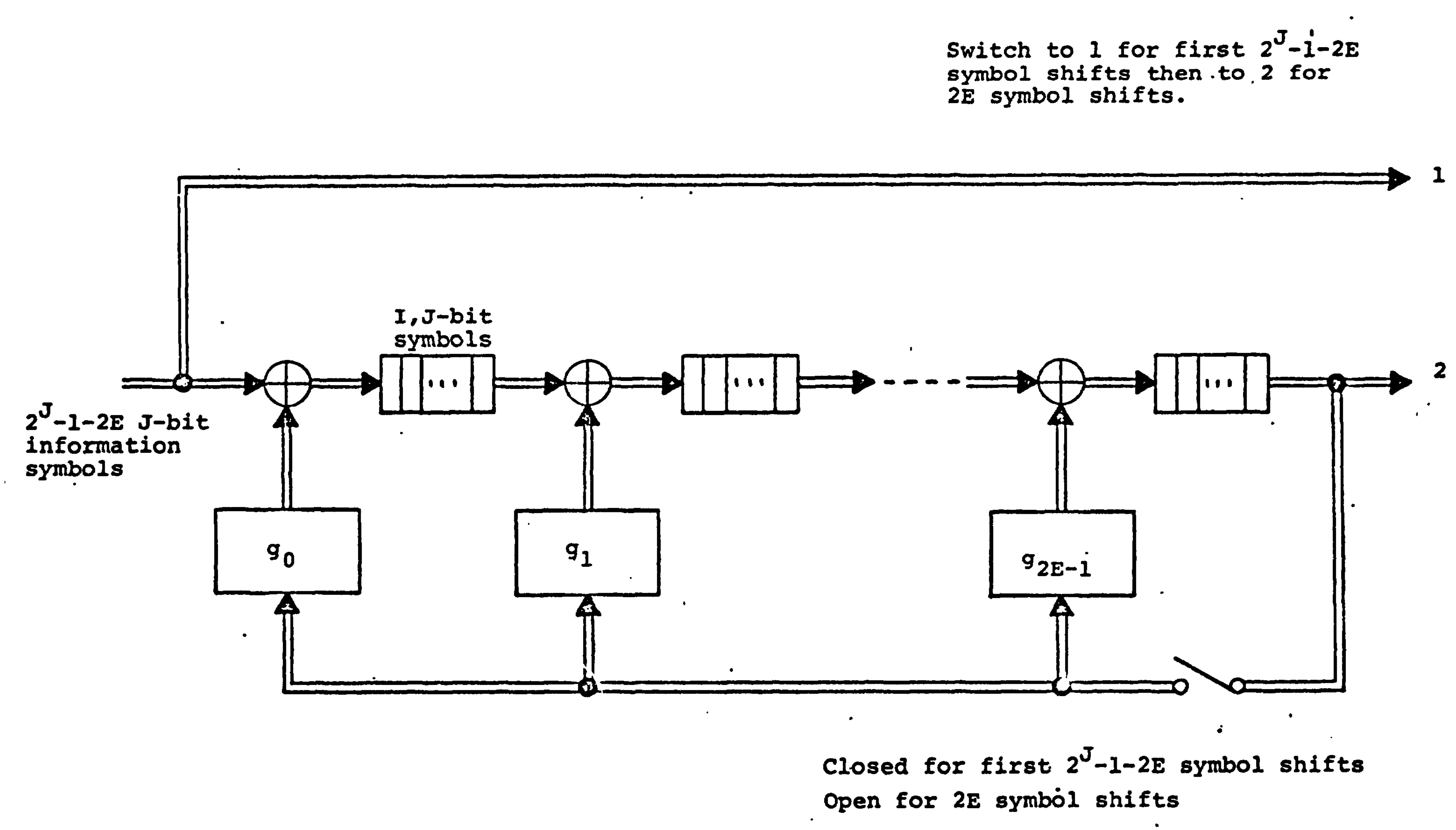


Figure 2.3.2 Outer Encoder and Interleaver.

where the coefficients, g_i, are from GF(2^J). In particular, if the field is generated by a primitive element α and α, α^2, α^3, ..., α^{2E} are roots of the code word polynomial, then

$$g(D) = \prod_{i=1}^{2E} (D-\alpha^i) \tag{2.3.2}$$

We will restrict our attention to this class of R-S codes throughout this report.

In an actual implementation the input and output are a sequence of binary symbols, so a serial-to-parallel operation must be performed at the input to the parity computation section and a parallel-to-serial operation must be performed before the outputs are fed to the convolutional encoder. A description of a hardware implementation of the encoder and interleaver is given in a later section. The important point is that the entire code array of Figure 2.3.1 does not have to be stored, only the 2EIJ parity bits need to be stored.

2.3.2 Unscrambler and Decoder Storage Implementation

The major storage requirement in this concatenated coding scheme is in the receiver unscrambler where the sequence of received R-S symbols must be grouped into R-S words and the decoded R-S symbols must be arranged so that they are presented to the data sink in the proper sequence. Figure 2.3.3 illustrates a method of implementing this unscrambling operation. This implementation operates as follows.

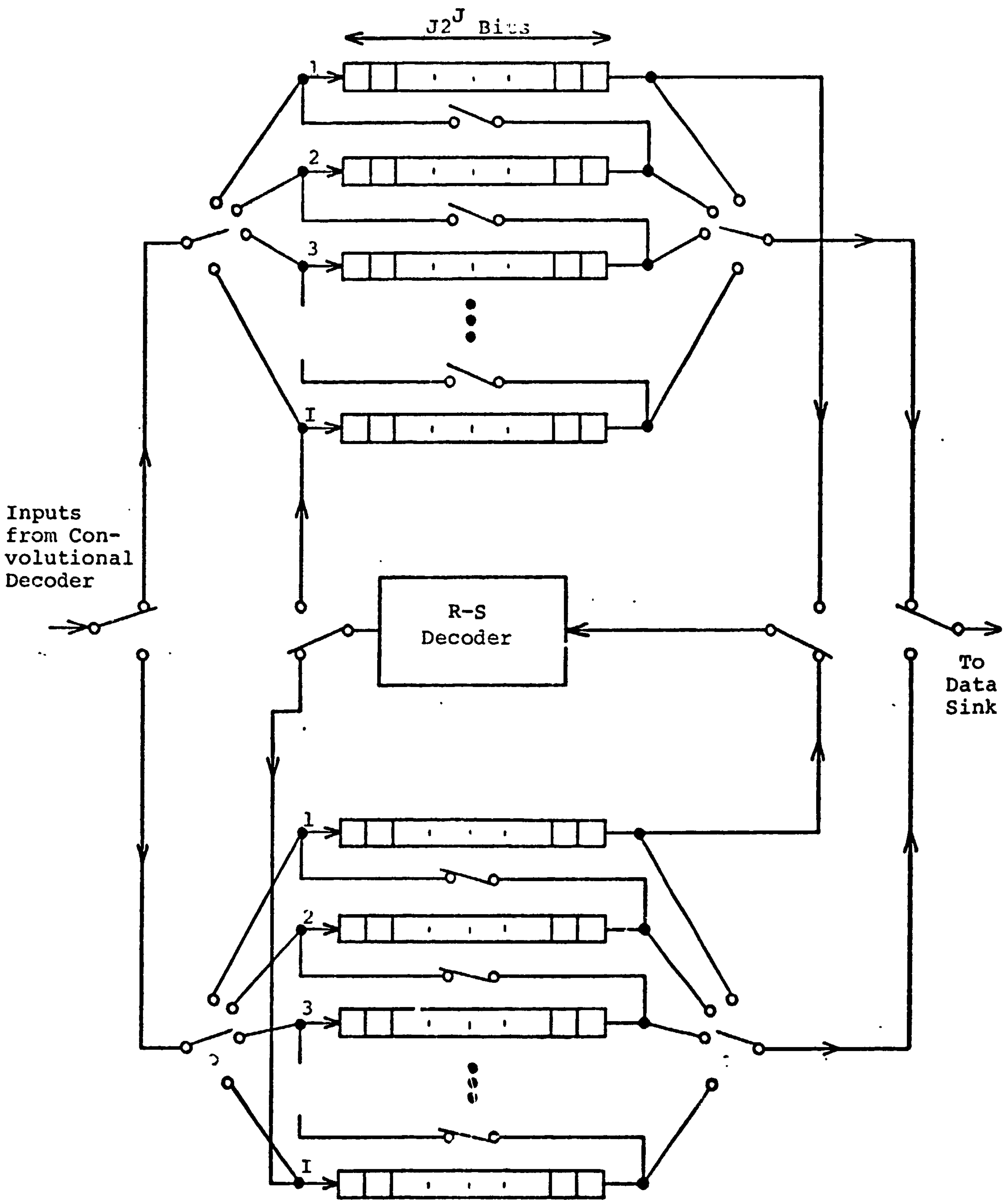


Figure 2.3.3. Unscrambler Implementation.

The first received symbol goes to the first register in the upper set of registers, the second received symbol to the second register, etc., until the I^{th} symbol is stored in the I^{th} register. Then the $(I+1)^{th}$ symbol is shifted into the first register and the procedure is continued until the registers are filled. Referring to the code array of Figure 2.3.1, it can be seen that this procedure puts the first I R-S words in the upper I registers. When these registers are filled, all of the switches are changed to their other position and the input symbols are shifted into the lower set of registers. Meanwhile the words in the upper register are shifted into the R-S decoder in the end-around manner shown and the corrected symbols are shifted back into the I^{th} register. After all the shifts have been completed, these registers contain a corrected version of the original I words. Then when the lower registers are filled, the positions of the switches are changed again and the words in the lower registers are shifted through the R-S decoder. The incoming symbols are shifted into the upper registers and the symbols shifted out are the decoded properly sequenced symbols.

This implementation has the advantage that the R-S decoder is independent of the interleaver. Other interleaving procedures could reduce the storage nearly by half, but at the cost of more complex control and staggered access to the R-S decoder. Investigation of these procedures has shown them to be less cost effective than the present one.

2.3.3 Reed-Solomon Decoding Procedure

A R-S decoder can be implemented in four steps:

1. Calculate the syndromes from the received sequence.
2. Use the Berlekamp Algorithm to find the error locator polynomial $\sigma(D)$.
3. Use a Chien Search to find the roots and hence the location of the errors.
4. Find the values of the errors.

The received word from the output of the inner decoder will be denoted

$$y(D) = \sum_{n=0}^{2^J-2} y_n D^n \qquad (2.3.3)$$

where y_{2^J-1-i}, $1 \leq i \leq 2^J-1$, represents the i^{th} received symbol. If α is a primitive element which generates the field, then the syndromes can be calculated by

$$\begin{aligned} S_i &= y\left(\alpha^{1+i}\right) \\ &= \left(\cdots \left(y_{N-1}\alpha^{1+i} + y_{N-2}\right)\alpha^{1+i} + y_{N-3}\right)\alpha^{1+i} \cdots + y_0\Big) \\ & \qquad\qquad\qquad\qquad\qquad\qquad 0 \leq i \leq 2E-1 \end{aligned} \qquad (2.3.4)$$

where N is the symbol block length of 2^J-1. Thus each syndrome can be calculated by adding each successive received R-S symbol into an initially empty register, multiplying the sum by α^{1+i}, and returning the result to the register awaiting the next received symbol. Figure 2.3.4 illustrates this procedure for the i-th syndrome.

The Berlekamp Iterative Algorithm for computing the error locator polynomial, $\sigma(D)$, from the syndromes is well documented by Berlekamp (Reference 11) and Massey (Reference 12). This algorithm is equivalent to synthesizing the minimum length shift register, over $GF(2^J)$, to generate $S_o, \ldots, S_{2E-1}$. The resulting tap coefficients are the coefficients of the error locator polynomial. We will use the notation and follow the block diagram given in Reference 10, Figure 6.7.4.

The Chien search determines whether a given symbol is in error by evaluating the polynomial

$$\sigma(D) = 1+\sigma_1 D+\ldots+\sigma_E D^E \qquad (2.3.5)$$

at all inverse values of the primitive field element α.
If

$$\sigma\left(\alpha^{-n}\right) = 1+\sum_{i=1}^{E} \sigma_i \left(\alpha^{-n}\right)^i \begin{cases} \neq 0, & \text{there is no error in the } n^{th} \text{ symbol.} \\ =0, & \text{there is an error in the } n^{th} \text{ symbol.} \end{cases}$$

$$n = 1, 2, \ldots, 2^J-1-2E$$

This search can be implemented as shown in Figure 6.7.5 of Reference 10.

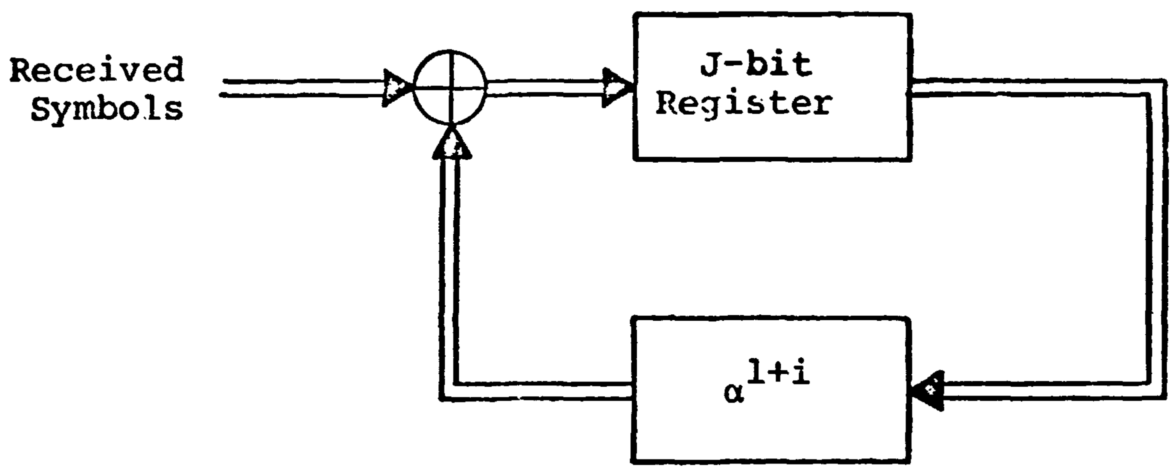


Figure 2.3.4 Procedure for Calculating the i^{th} Syndrome.

After the error locations have been located with the Chien search, the error values must be calculated. If less than or equal to E errors have occurred, the error values are given by the formula (Reference 10)

$$V_n = -\frac{A(\alpha^{-n})}{\sigma'(\alpha^{-n})}, \quad n=n_1, n_2, \ldots, n_E \tag{2.3.6}$$

where n_i is the location of the i-th error

$$\sigma'(D) = \sigma_1+\sigma_3 D^2+\sigma_5 D^4+\ldots+\sigma_F D^{F-1} \text{ where } F=\begin{cases} E, & \text{if } E \text{ odd} \\ E-1 & \text{if } E \text{ even} \end{cases} \tag{2.3.7}$$

and

$$A(D) = \Big[S(D)\sigma(D)\Big]_0^{E-1} = S_0 + (S_0\sigma_1+S_1)D+(S_0\sigma_2+S_1\sigma_1+S_2)D^2$$

$$+\ldots+(S_0\sigma_{E-1}+S_1\sigma_{E-2}+\cdots+S_{E-1})D^{E-1} \tag{2.3.8}$$

2.4 Part Software and Part Hardware Decoder Implementation

Several parts of the concatenated decoder are ideally suited to hardware implementation. As noted previously, the hardware implementation of constraint length 7 and 8 Viterbi decoders is relatively straightforward at speeds of less than 100 Kbps. Also, the unscrambler of Figure 2.3.3 can be easily implemented in hardware, but it would require a large amount of storage to implement in software. So these operations should clearly be done in hardware.

The R-S decoder can be efficiently implemented entirely in hardware or in part hardware and part software. The most efficient implementation will depend on the required speed and the code parameters.

Since the performance curves indicate that high rate R-S codes should be used, the slowest parts of the R-S decoding are steps 1 and 3 which have to be performed for each received symbol. This indicates that these steps should be perfomred in hardware. However, the interfacing problem in going from a software step 2 to a hardware step 3 and then back to a software step 4 may make it desirable to perform step 3 in software also.

To estimate the speeds of these three steps in the R-S decoder, we wrote a computer program to perform these steps. One of the problems in writing such a program is to find efficient ways of storing, adding, and multiplying field elements. Field elements over $GF(2^J)$ can be represented as powers of a primitive field element α or as (J-1)- degree polynomials over GF(2). In this program we represented the field elements by J-bit integers with the bits corresponding to the coefficients in their polynomial representation. Field addition is accomplished with a bit-by-bit exclusive-OR operation.

Field multiplication and division are performed using field log and anti-log tables. The log table lists the corresponding power of α for the integer representations of the 2^J-1 non-zero field elements and the anti-log table lists the integer field element representations for the powers of α. With these tables multiplication/division of two non-zero field elements is accomplished by adding/subtracting their logs modulo (2^J-1) and looking up the resultant in the anti-log table.

Appendix A gives a listing of a Fortran and an assembly language version of this program. The decoding speeds of the various steps in the assembly language program are given in Table 2.4.1 for two sets of R-S code parameters.

These times are for the LINKABIT, Digital Scientific META-4 Computer with a one microsecond core memory cycle. Each time is based on the time to decode three arbitrarily chosen sets of E error locations and values. For the cases timed there was less than a 3% variation in these times. Table 2.4.1 lists the largest of the three times.

This table shows that, at least for the two R-S decoders timed, a serial software implementation of steps

Step	2^8-Symbol, 16-Error-Correcting R-S Code		2^7-Symbol, 8-Error-Correcting R-S Code	
	Time Required To Decode 1000 Blocks in Seconds	Speed In Channel Kbps	Time Required To Decode 4000 Blocks in Seconds	Speed In Channel Kbps
2	47.0	43.4	50.9	69.8
3	148.3	13.7	156.3	22.7
4	35.3	57.8	38.1	93.2
2 and 4 serially	82.3	24.8	89.0	40.0

Table 2.4.1 Software R-S Decoder Speeds.

2, 3, and 4 is too slow. If the Chien search (step 3) is performed in hardware and two minicomputers are used for steps 2 and 4, the decoding speed is limited by the speed of the Berlekamp Algorithm (step 2). Table 2.4.1 indicates that such an implementation would be satisfactory with lower speed requirements or for codes with smaller guaranteed error-correcting abilities.

This program could be speeded up by perhaps as much as a factor of 10 by using micro-programming techniques. If this were done, steps 2, 3, and 4 of the Berlekamp Algorithm could probably be serially implemented at up to 100 Kbps for the 2^7-symbol 8-error-correcting code. However, a hardware implementation appears desirable for the more powerful 2^8-symbol 16-error-correcting code.

2.5 Hardware Implementation

The present discussion on the hardware implementation will be limited to a system with a 2^8-symbol* 16-error-correcting R-S code and an interleaver length of 16. In section 2.2 it was shown that for this alphabet size and desired range of error probabilities, 16 is the optimum number of correctable errors. The computer simulation also showed that in this case an interleaving length of 16 was sufficient for probabilities of error down to 10^{-8}. Here we will show that this system can be hardware implemented at a reasonable cost. The design principles are the same for systems with different high rate, low speed R-S decoders.

First, we discuss the hardware implementation of some of the basic field operations. Then we present an outline of a hardware implementation with an estimate of the number of integrated circuit chips required to accomplish the operations.

2.5.1 Hardware Implementation of Field Operations

As in the software implementation, let the $GF(2^8)$ field elements be represented by polynomials of degree less than 8 in α. That is, a field element Y is represented as

---

* This is a particularly convenient field size since then the field elements can be stored in 8-bit shift registers.

$$Y = \sum_{i=0}^{7} Y_i \alpha^i \qquad (2.5.1)$$

where the Y_i coefficients are binary numbers. Also, in order to obtain specific circuits for performing field multiplication, let the $GF(2^8)$ field be generated by a field element α with a primitive polynomial

$$M(D) = 1 + D^2 + D^3 + D^4 + D^8 \qquad (2.5.2)$$

The only criterion used in selecting this primitive polynomial is that it have minimum weight which in this case is 5.

One method of multiplying two non-zero field elements is to look up their logarithms in a log table, add the logs modulo 255, and look up the result in an anti-log table. Each log and anti-log table look-up can be implemented with a 256 x 8 read-only memory (ROM) and the addition can be implemented with two chips. In general, to multiply two arbitrary field elements a test would have to be made to determine if either were zero and, if this were the case, the output would be set to zero. Thus, excluding control circuitry, 7 chips are required*.

---

*This is reduced when a variable element is multiplied by a fixed element (such as in polynomial evaluation where the fixed element is a polynomial coefficient) since then we can simply store the logarithm of the fixed element rather than the element itself, thus avoiding one ROM, and if the fixed element is non-zero, one zero test chip.

Another method (Reference 11) of multiplying two field elements, U and V, is illustrated in Figure 2.5.1. Initially the two field elements to be multiplied are stored in the U and V registers and the Z register is set to zero. The U register is wired to multiply by α, the V register is a storage register which can be shifted to the right, and Z is an accumulator register. The multiplier operates as follows. Depending on the lowest bit of V, U is either added or not added into Z. Then the U and V registers are shifted and the process is repeated. After 8 steps Z contains $\sum_{i=0}^{7} V_i (U\ \alpha^i)$, the desired product.

Figure 2.5.2 gives an implementation of this field multiplication procedure. In this and the proceeding implementation diagrams, L denotes low, H high, and X irrelevant. Excluding control circuitry, this implementation requires 8 chips. However, the chips required here are less costly than those in the previous implementation.

The best way of obtaining the inverse of a field element is to look up the answer in a table containing the 2^8-1 inverses. This can be implemented with one 256 x 8 ROM.

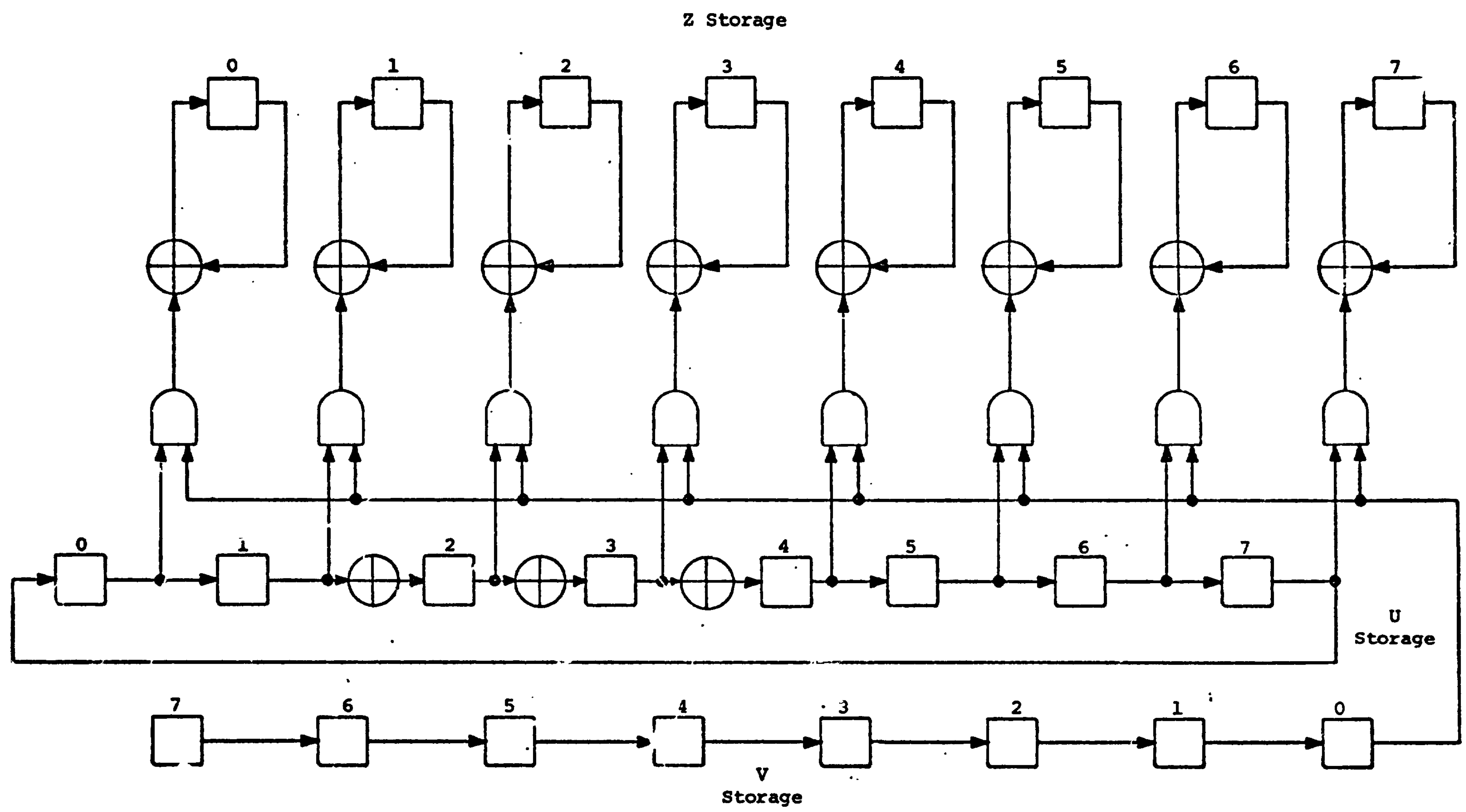


Figure 2.5.1 Alternate Field Multiplication Procedure.

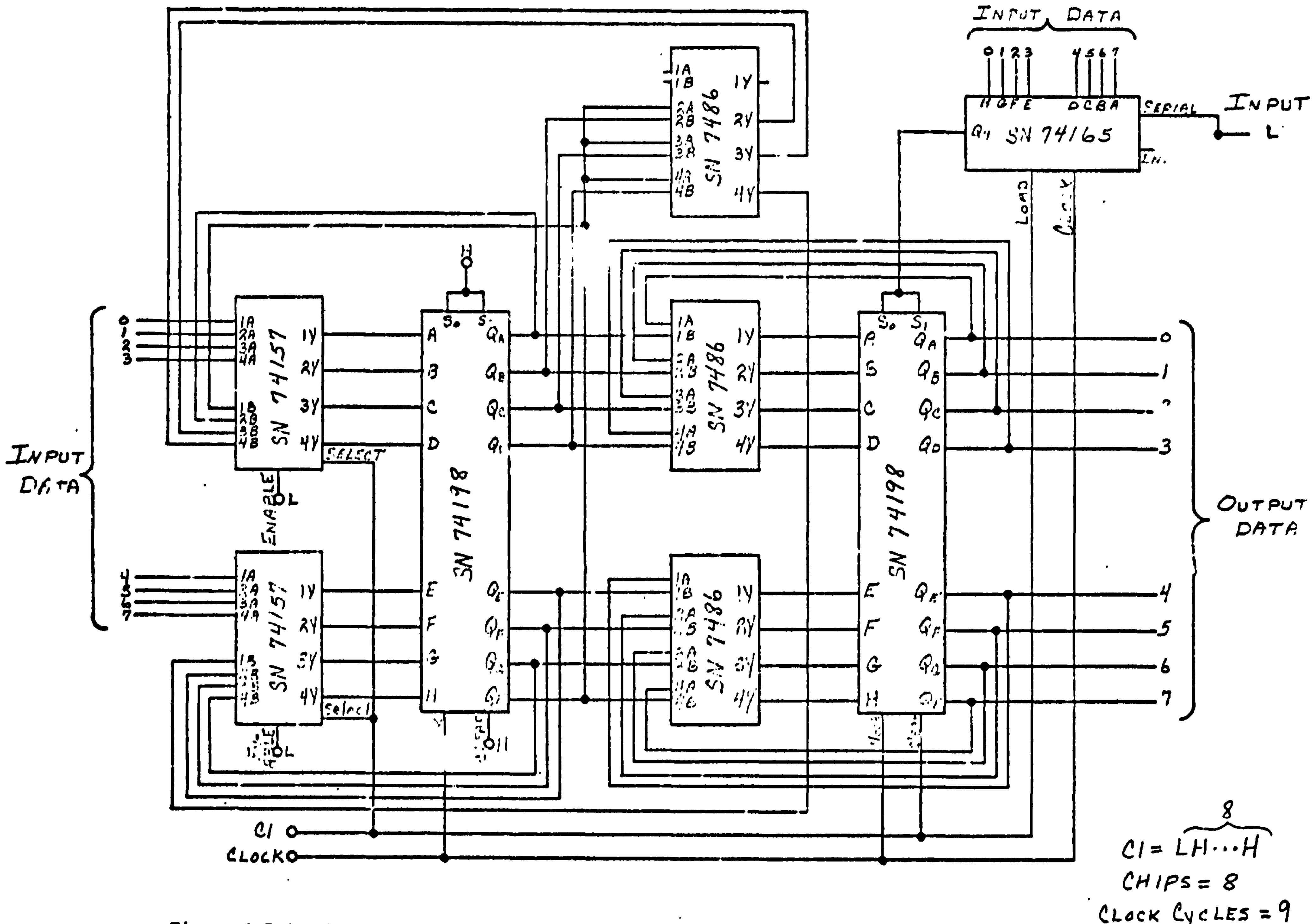


Figure 2.5.2 Alternate Field Multiplication Implementation.

2.5.2 Hardware Encoder and Interleaver Design

In section 2.3.1 we described the general procedure for implementing the encoder and interleaver. Figure 2.5.3 gives an outline of a hardware implementation of this procedure. Random-access memories (RAM's) provide the parity symbol storage and a read-only memory (ROM) provides the storage for the logs of the generator polynomial coefficients. The main difference between this implementation and the procedure shown in Figure 2.3.2 is that here the multiplications are performed in series instead of in parallel as indicated in Section 2.3.1. That is, for each input symbol, 32 cycles through this circuitry are required to update all of the parity symbols for that R-S word. Then the RAM selects the next set of 32 parity symbols and the same procedure is repeated for the next input symbol. This serial computation procedure, of course, takes longer than the parallel procedure, but it is fast enough to obtain the required coding speeds and it has far fewer parts.

Above each block in this diagram is an estimate of the number of TTL chips required to accomplish the operation. The composite RAM shown requires four 1024 x 1 RAM chips and must be clocked twice to obtain the desired 8-bit output. The field multiplication is performed using the logarithmic procedure described in the previous section. However, the complexity of this multiplier is reduced a little by storing the logarithms of the g_i coefficients instead of their polynomial representations. If any coefficient is zero,

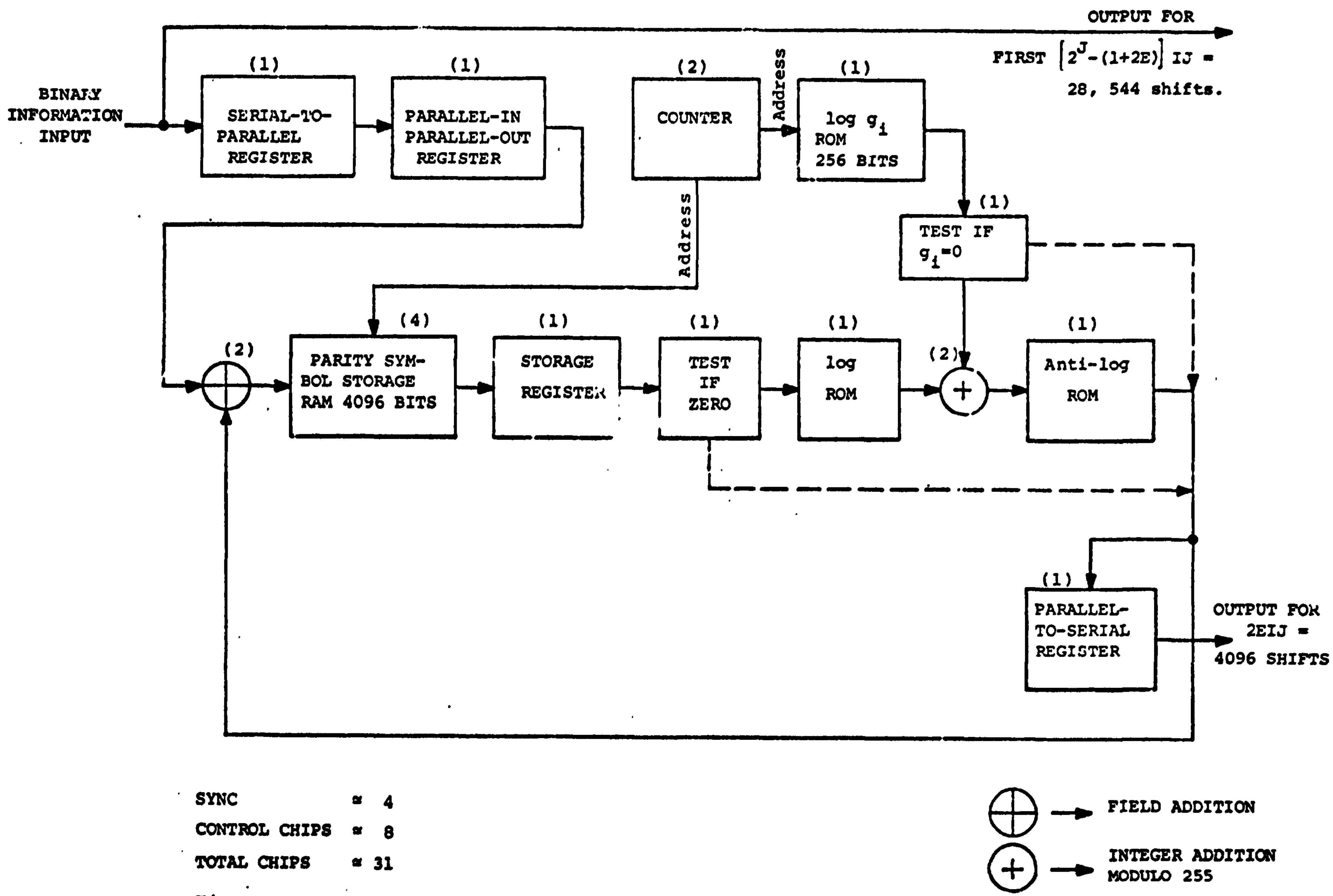


Figure 2.5.3. Outer Encoder and Interleaver Implementation.

THIS PAGE INTENTIONALLY BLANK

the value 255 can be substituted since the largest logarithm is 254. The dotted lines indicate that if either multiplier input is zero, the output is zero.

This diagram does not include the control or synchronization circuitry. It is estimated that 8 and 4 chips, respectively, are required to accomplish these operations.

2.5.3 Synchronization Implementation

As described in Section 2.3, the Viterbi decoder provides inner code synchronization and phase ambiguity resolution and the IJ = 128 bit sequence, consisting of 16 superchannel symbols, is used to obtain block code array synchronization. At the moderate data rates required here, the code array synchronization can be implemented with a simple correlation detector. That is, for each received superchannel bit the detector correlates the 128-bit received sequence, terminating at that bit, with a locally generated copy of the synchronization sequence and compares the output with a threshold to determine the starting bit of the code array. The most recent 128 superchannel bits can be stored in a RAM, the synchronization sequence can be generated with 2 chips, and the correlator, consisting of an exclusive-OR circuit and a counter, can be implemented with a little over 2 chips. Adding a few chips for control circuits, a total of about 8 chips are required.

This, of course, requires that for each bit time ($\geq$ 10 μsec.) the locally generated synchronization sequence be shifted and modulo-2 added to the stored most recently received 128 bits. Thus, the synchronization sequence must be shifted at a speed of up to 12.8 MHz, which is well within the capabilities of the TTL logic.

2.5.4 Hardware Unscrambler Implementation

For the J=8, I=16 system being considered, the unscrambler of Figure 2.3.3 requires 2^{16}= 65,536 bits of storage. The best way of implementing this is to use 16 4096 x 1 MOS RAM's. Dynamic MOS shift registers could also be used, but they would have to be recirculated at the lower data rates. Using the MOS RAM's, 16 chips are required for the storage requirements and it is estimated that an additional 14 chips are required for the control and rather formidable addressing circuitry. Thus a total of 30 chips are required.

2.5.5 Hardware Reed-Solomon Decoder Design

A sketch of the overall design of a R-S decoder is shown in Figure 2.5.4. Typically the decoder will be computing the syndromes for one word while the remaining decoding steps are performed for the previous word. The Chien searcher checks each symbol to see if an error has occurred in the symbol about to be shifted out of the buffer. If so, the error value is computed and the symbol is corrected.

A timing diagram of the R-S decoding operation is given in Figure 2.5.5. The lines in this figure indicate the relative amounts of time and the sequence of operations in the Reed-Solomon decoding procedure.

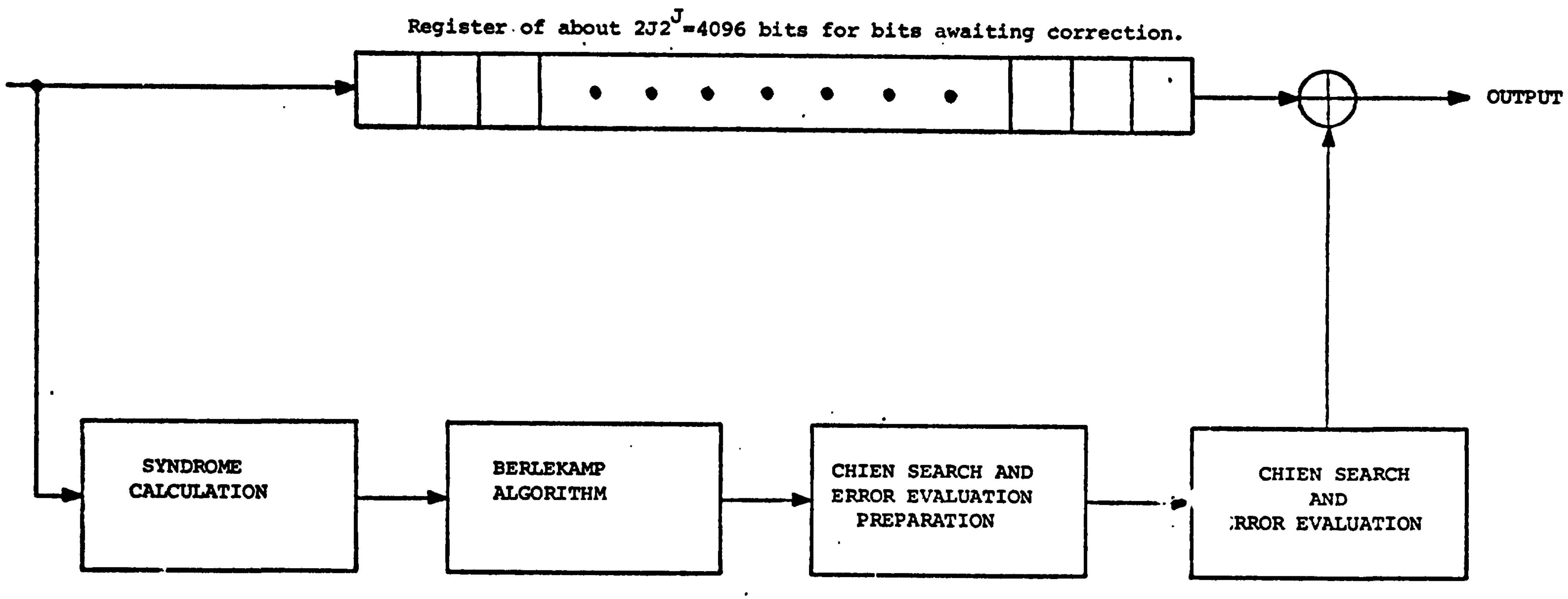


Figure 2.5.4. Reed-Solomon Decoder Block Diagram.

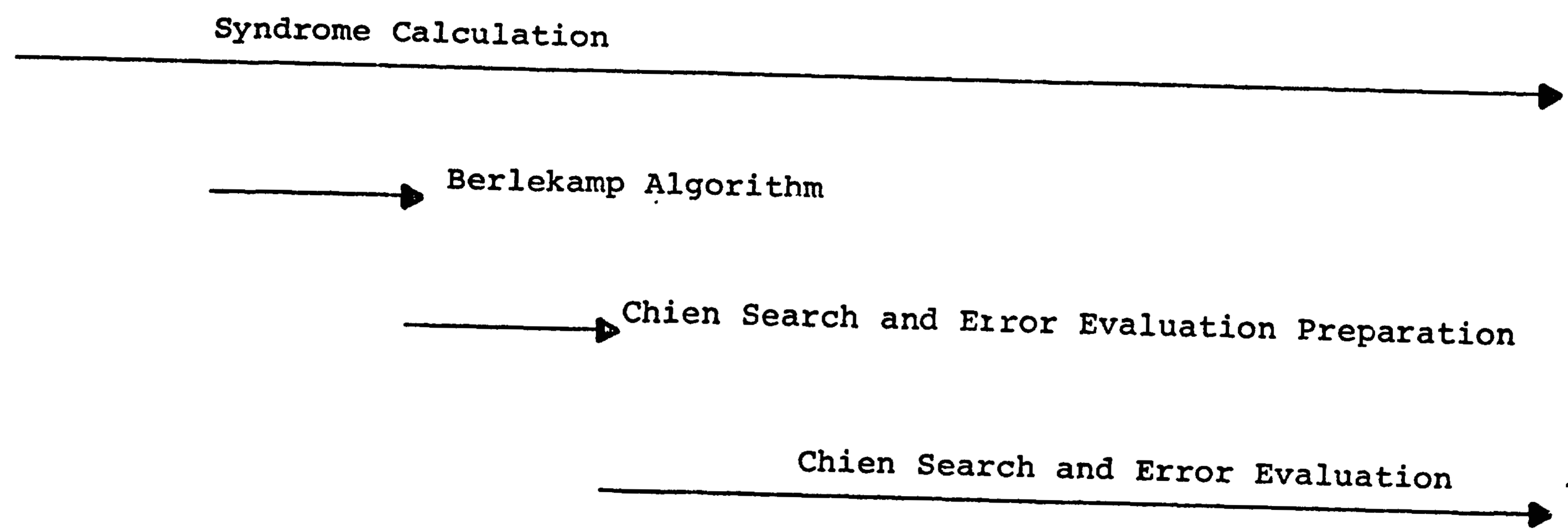


Figure 2.5.5 Reed-Solomon Decoder Timing Diagram.

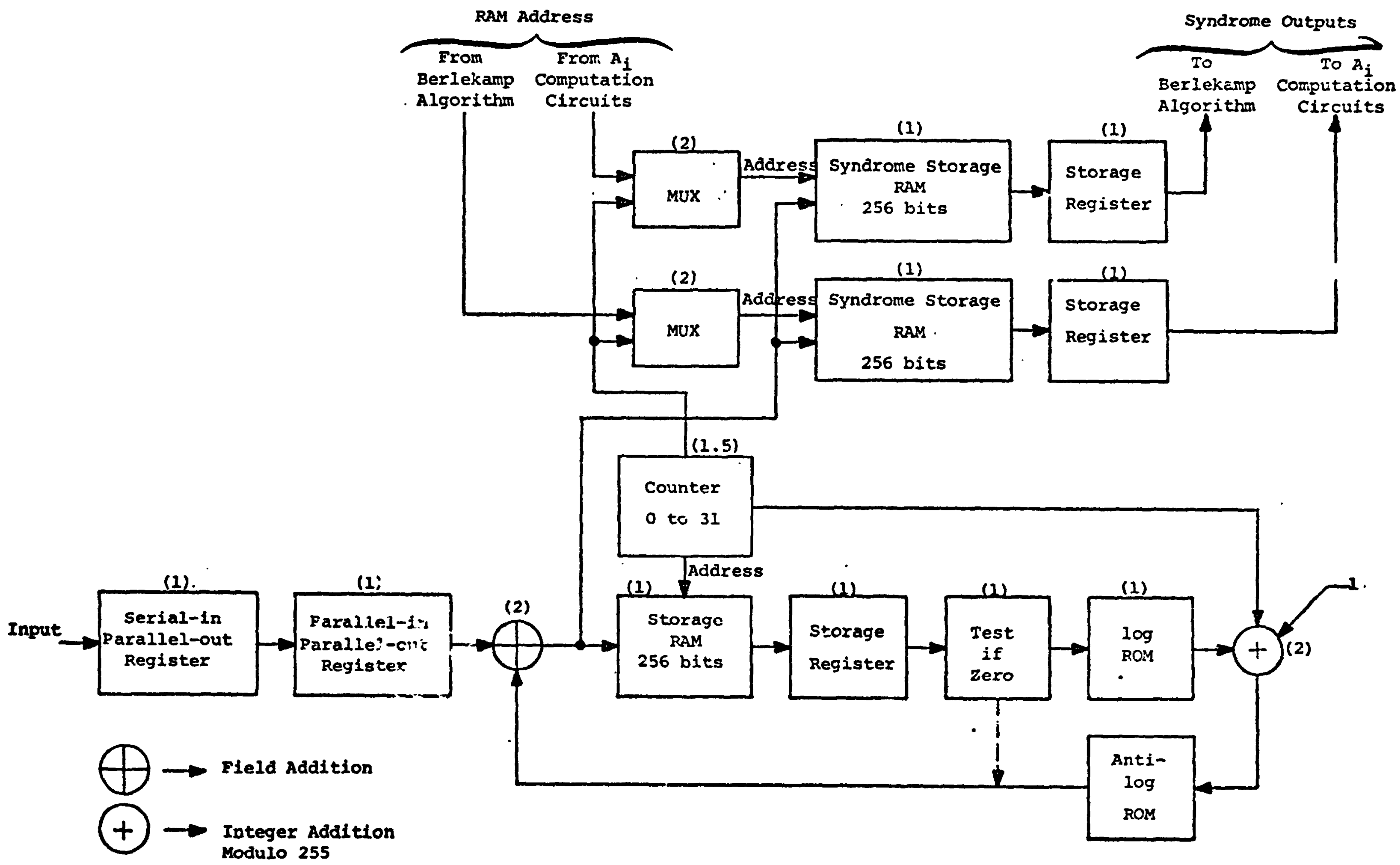
2.5.5.1 Syndrome Calculation

As with the encoder implementation, the number of chips required to implement the syndrome calculation can be greatly reduced by using a serial instead of a parallel implementation. Figure 2.5.6 shows a serial implementation of this procedure. Referring to Figure 2.3.4, the counter of Figure 2.5.6 generates the logs of the α^i elements and the lower RAM contains the storage for the syndromes being calculated. During the period immediately following the first received symbol of the word, the feedback is removed and in 32 serial steps the first term of each of the syndromes is written into the lower RAM. When the remaining symbols in the word are received, the feedback is used to modify the syndromes as shown in Figure 2.3.4. Again 32 steps per received symbol are required to modify all of the syndromes. On the last series of modifications, i.e., after the last symbol of the word is received, the syndromes are also stored in the upper RAM's for use in the other decoding steps.

The estimated number of TTL chips required to implement the various steps and the control circuits are shown.

2.5.5.2 Berlekamp Algorithm Implementation

Reference 10 provides a good description of the Berlekamp Iterative Algorithm. Basically the algorithm synthesizes the shortest length shift register which will generate the syndrome sequence. The resulting tap connections are the



Control Chips $\approx$ 6.5

Total Chips $\approx$ 27

Figure 2.5.6 Syndrome Calculation.

coefficients of the error locator polynomial. As described in Reference 10 and illustrated in Figure 6.7.4 of Reference 10, at each iteration the algorithm computes the discrepancy, d_n, between the next syndrome and the next output of the present shift register. If this discrepancy is not zero, a new set of tap connections is generated.

Figure 2.5.7 gives a simplified block diagram of the algorithm and Figures 2.5.8 and 2.5.9 outline an implementation of the two main parts of this block diagram.

In Figure 2.5.8 the R1 RAM contains the present set of shift register tap connections and the d_n register accumulates the terms of the next discrepancy as indicated.

The diagram of Figure 2.5.9 illustrates the operation of the main processor in the notation of Figure 6.7.4 of Reference 10. At each iteration this processor checks to see if the next discrepancy is zero. If it is, this processor merely shifts the words in the R3 RAM one address location and inserts a "0" symbol. If the next discrepancy is not zero, a new set of shift register tap sequences must be computed. This is accomplished by modifiying each of the 16 words in the RAM's as shown and then shifting the words in the R3 RAM one address location and inserting a "0" or a "1" symbol, depending on the polarity of $n-2\ell_n$. Also if $d_n \neq 0$ and $n \geq 2\ell_n$, ℓ_n and d* must be updated as indicated.

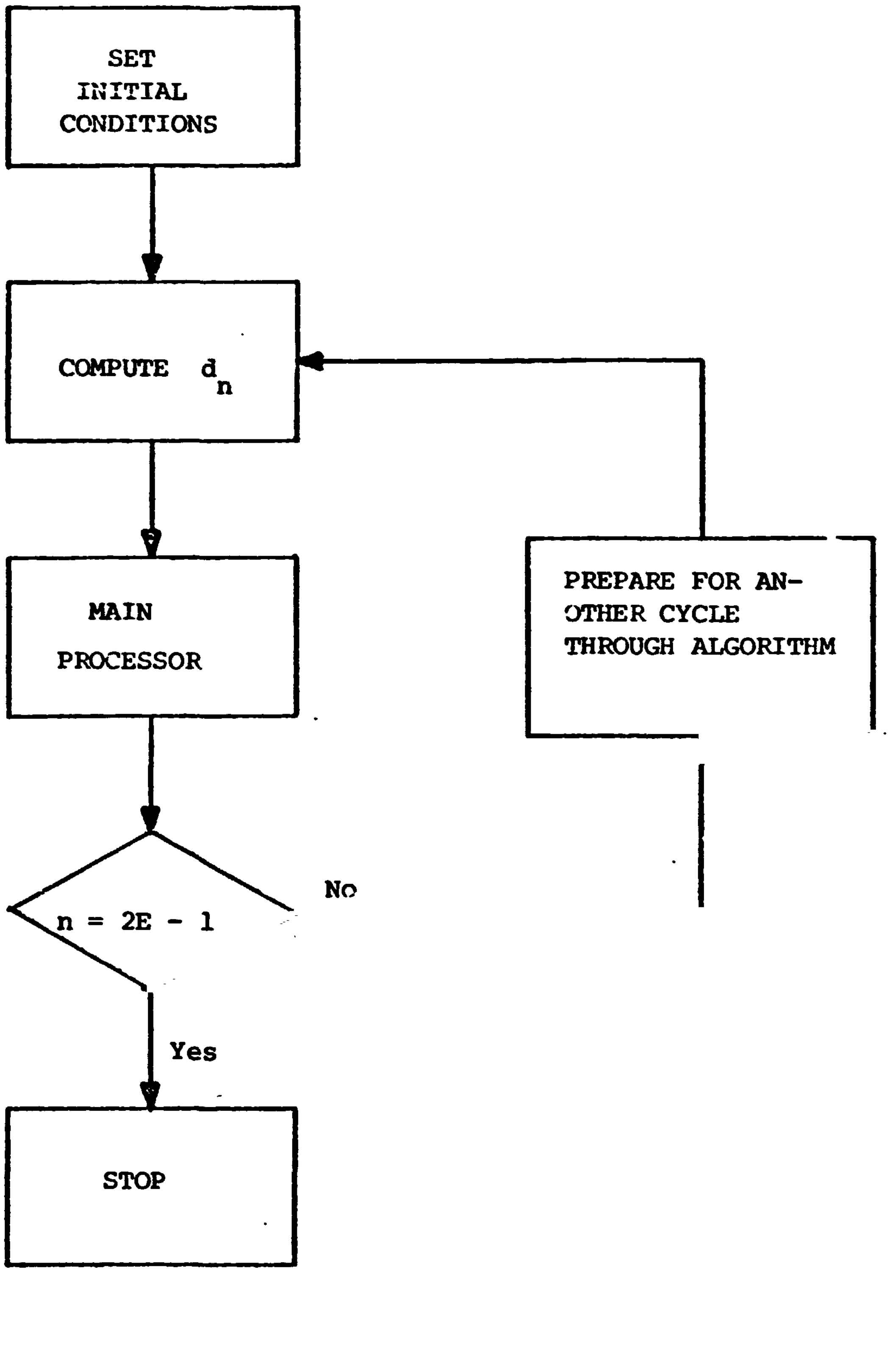
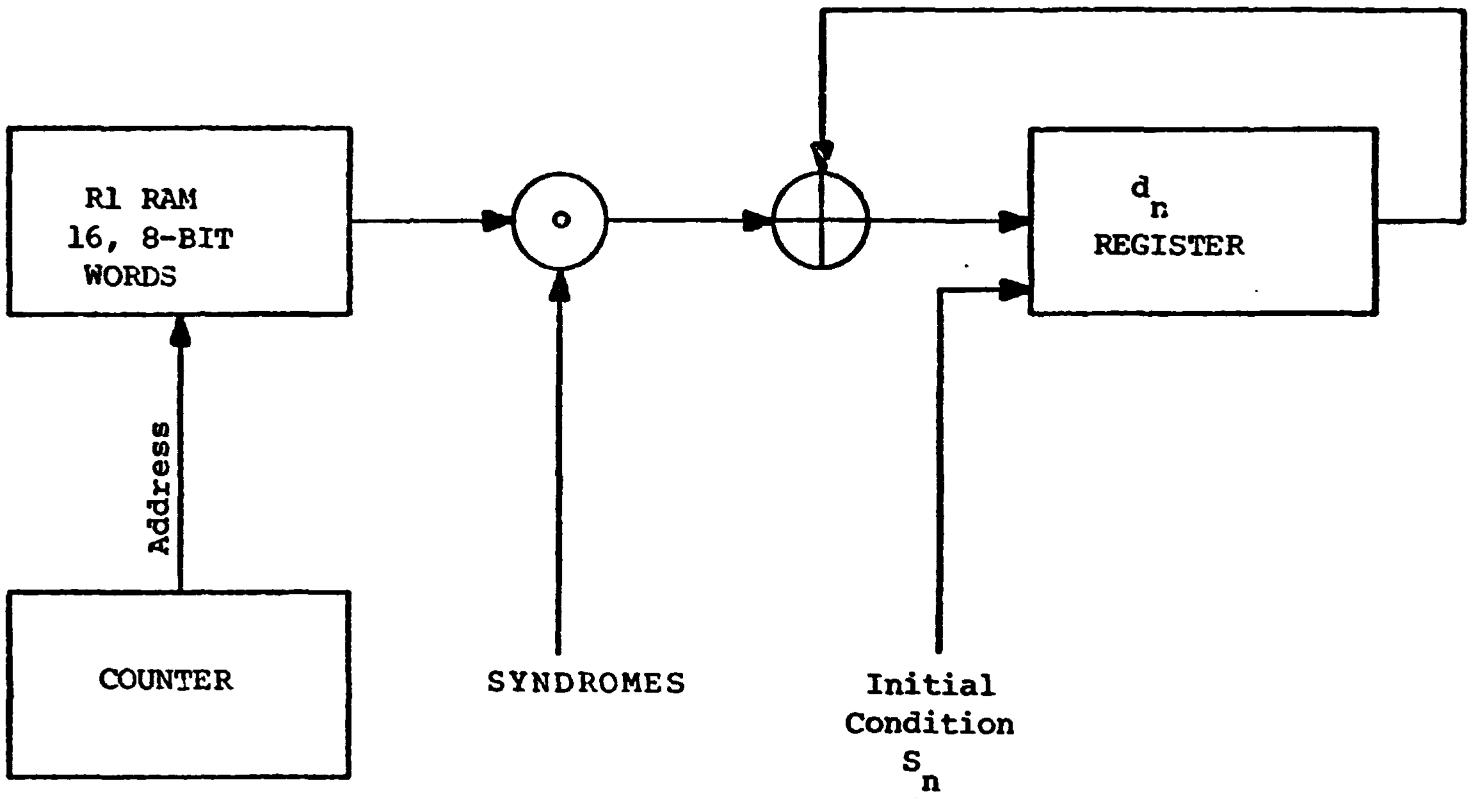


Figure 2.5.7. Berlekamp Algorithm Block Diagram.



$$d_n = S_n + \sum_{i=0}^{\min(n-1,15)} (R1)_i \, S_{n-i-1}$$

Figure 2.5.8. d_n Calculation Procedure.

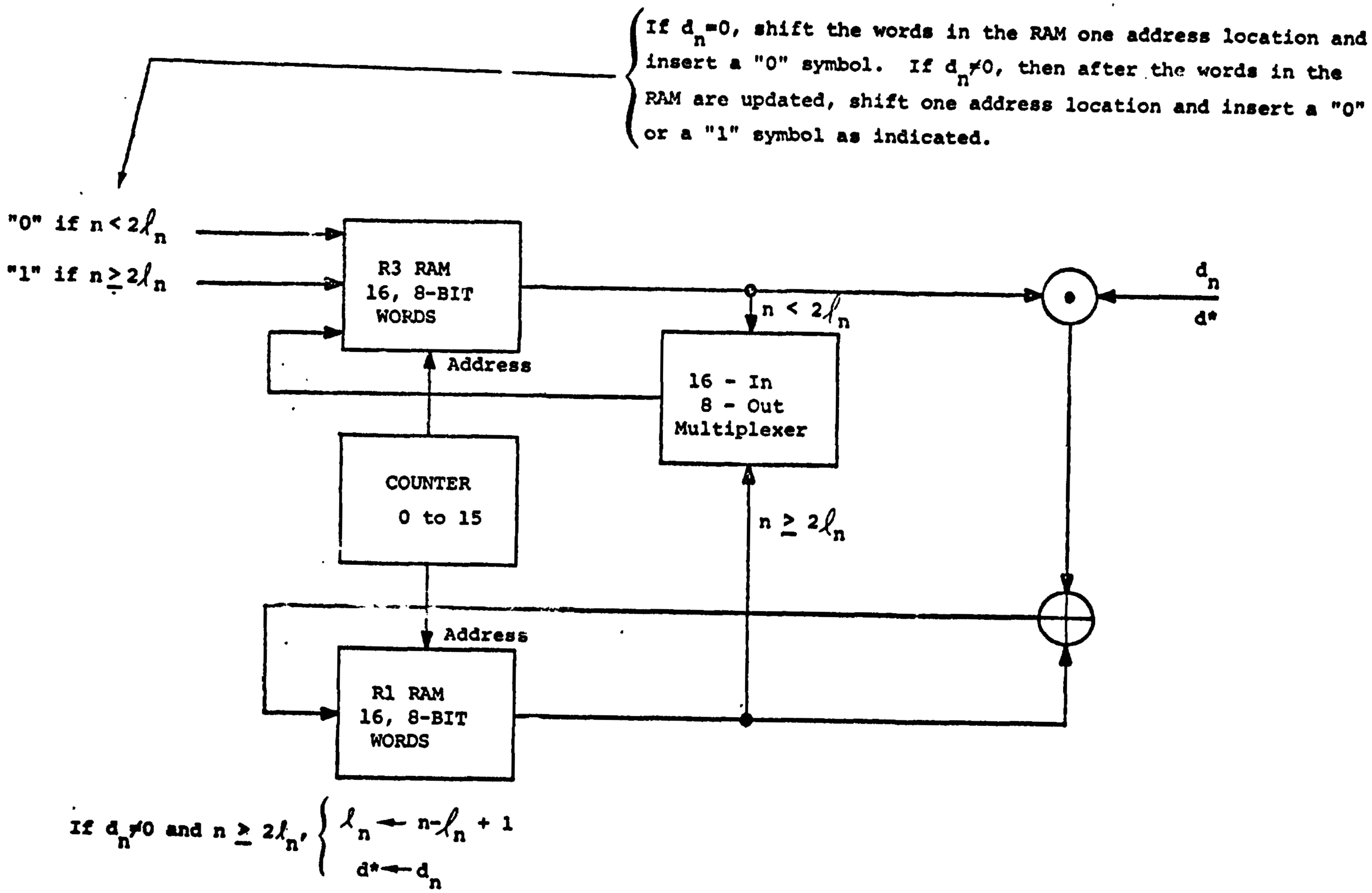
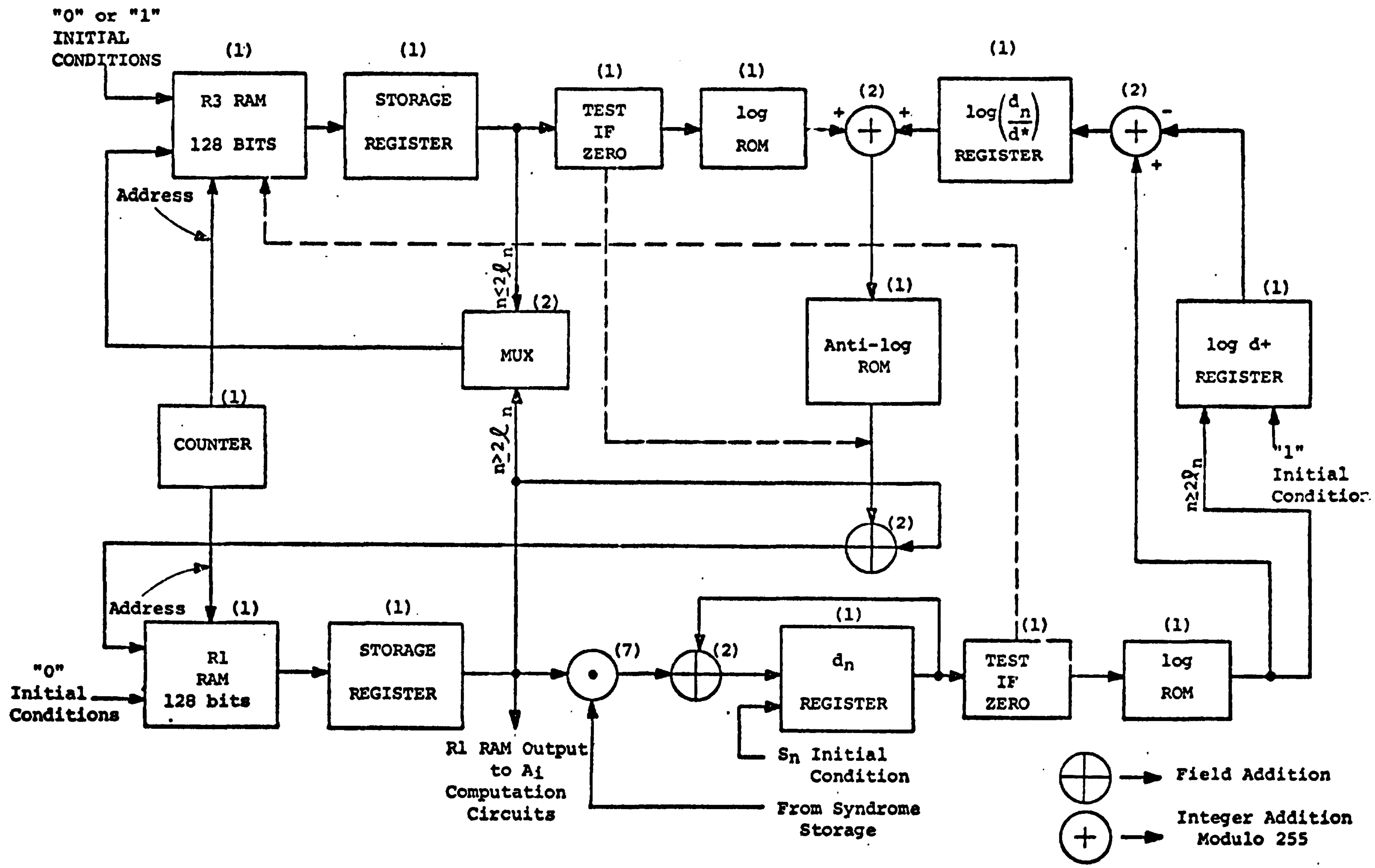


Figure 2.5.9. Block Diagram of the Main Processor of the Berlekamp Algorithm.

Figure 2.5.10 sketches an implementation of this algorithm. This implementation uses the procedures outlined in Figures 2.5.8 and 2.5.9 and adds circuitry to implement some of the other operations.

The dotted line from the $d_n=0$ tester indicates that when $d_n=0$, control is shifted to the R3 RAM as described in Figure 2.5.9. The other dotted lines indicate that, as before, when a multiplier input is zero, the output is set to zero.



Chips for control circuits ≈ 12

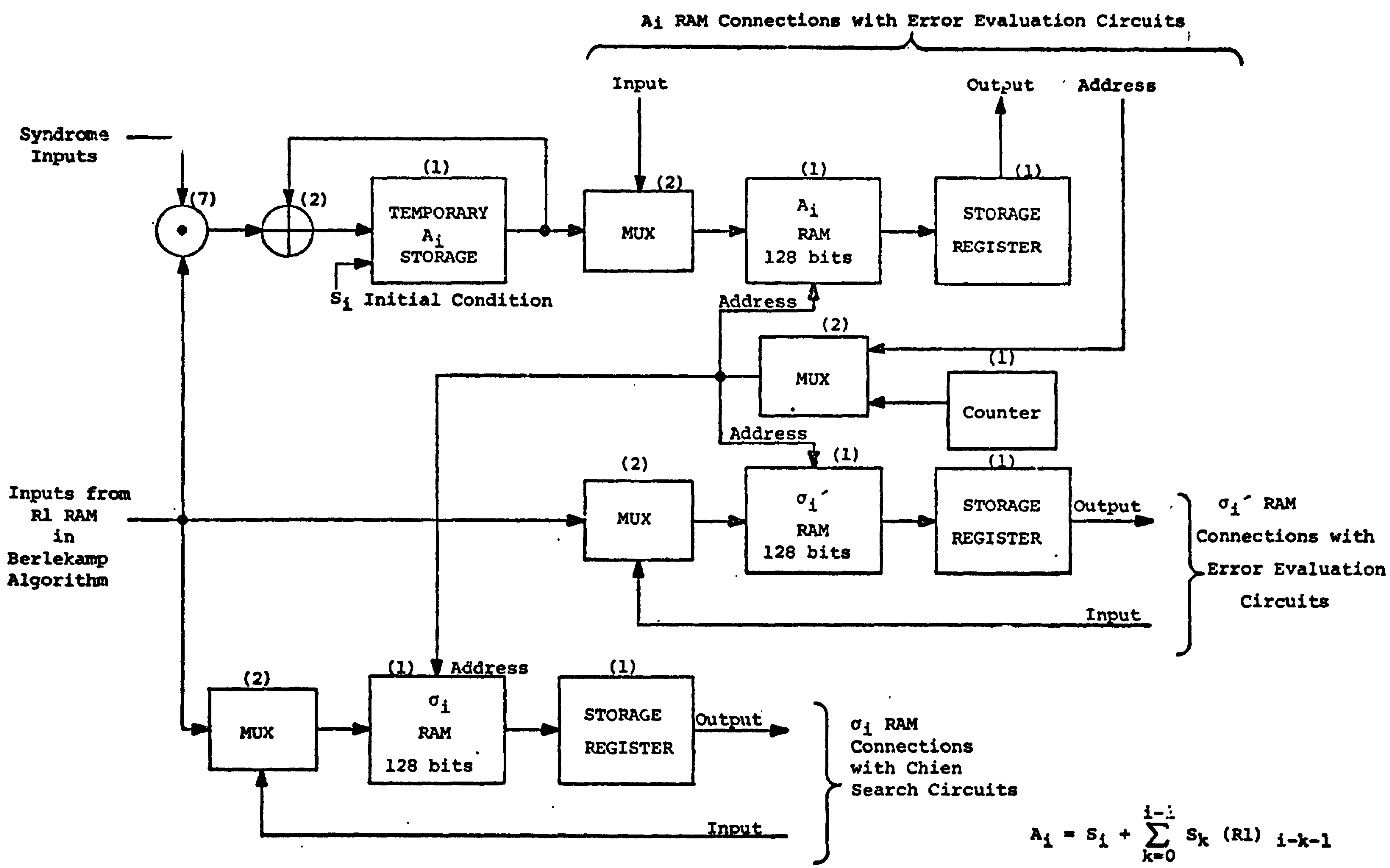
Total chips ≈ 42

Figure 2.5.10 Berlekamp Algorithm Implementation.

2.5.5.3 Chien Search and Error Evaluation Preparation

The Chien search and error evaluation preparation step stores and, when necessary, computes the coefficients of the A, σ, and σ' polynomials so that they can be used efficiently in the Chien search and error evaluation procedure. The coefficients of the σ polynomial, σ_i, are computed with the Berlekamp Algorithm and, as shown in Equation 2.3.7, the coefficients of the σ' polynomial, σ'_i, are equal to σ_{i+1} for i even and zero otherwise. So this step merely stores these coefficients so that they can be readily accessed by the Chien search and error evaluation circuits.

The coefficients of the A polynomial must be computed. They can be computed directly from the formula (2.3.8) or their calculation can be incorporated into the Berlekamp Algorithm (Reference 10). In this case, the direct approach appears to be less complex to implement. Figure 2.5.11 gives an outline of an implementation using this approach. This implementation accumulates the sum defining each coefficient in the temporary A_i storage register and then stores the result in the A_i RAM.



$$A_i = S_i + \sum_{k=0}^{i-1} S_k \,(R1)\; i\text{-}k\text{-}1$$

$$\sigma_i' = \begin{cases} (R1)_i & , i \text{ even} \\ 0 & , i \text{ odd} \end{cases}$$

$$i = 0, 1, \ldots, 15$$

Chips for Control Circuits $\cong$ 6

Total Chips $\cong$ 31

Figure 2.5.11 Chien Search and Error Evaluation Preparation.

2.5.5.4 Chien Search and Error Evaluation

As described in section 2.3.3 the Chien search determines whether a given symbol is in error by computing $\sigma(\alpha^{-n})$. If this quantity is nonzero, the n^{th} symbol is said to be correct. Otherwise an error of value $A(\alpha^{-n})/\sigma'(\alpha^{-n})$ is said to have occurred in that symbol.

Figure 2.5.12 gives an implementation of this procedure. This implementation performs the Chien search and evaluates the A and σ' polynomials in parallel, first for $n = N-1$, then for $n = N-2$, and so forth, where $N = 2^J-1=255$. In the first step the circuitry accumulates

$$\sigma\left(\alpha^{-(N-1)}\right) = 1 + \sum_{i=0}^{15} \sigma_{i+1}\, \alpha^{i}\alpha$$

$$A\left(\alpha^{-(N-1)}\right) = \sum_{i=0}^{15} A_i\, \alpha^i$$

and

$$\sigma'\left(\alpha^{-(N-1)}\right) = \sum_{i=0}^{15} \sigma_i'\, \alpha^i$$

in the σ, A, and σ' storage registers, respectively. Then the NOR gate checks to see if the first received symbol, y_{N-1}, is correct. That is, it checks to see if $\sigma(\alpha^{-(N-1)})$ is nonzero. If so, the output AND gate produces a sequence of 8 zero bits. If the NOR gate output is high, an error is

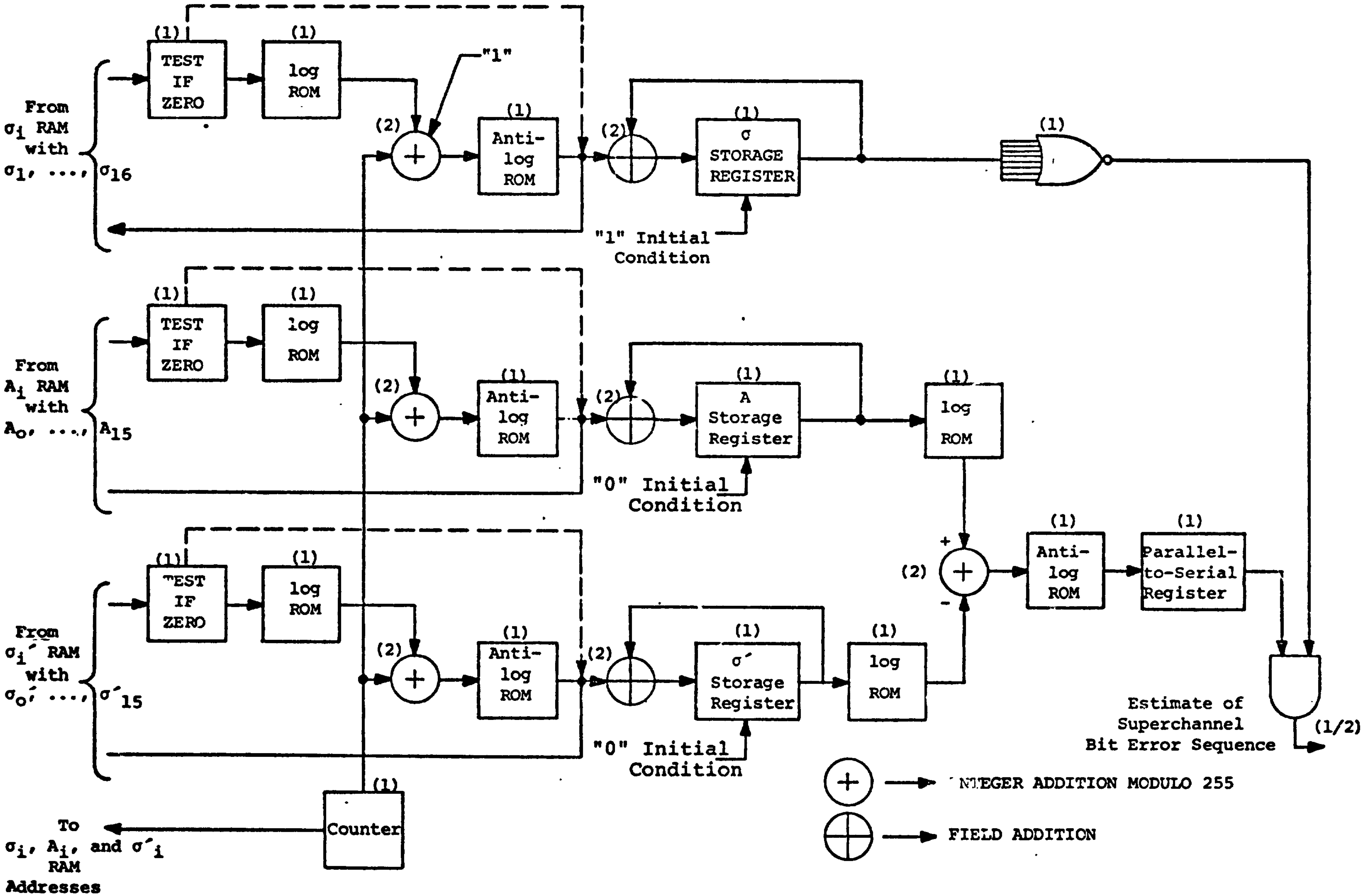


Figure 2.5.12 Chien Search and Error Evaluation Procedure.

Chips for Control Circuits $\cong$ 7.5
Total Chips $\cong$ 40

indicated. In this case $A(\alpha^{-(N-1)})/\sigma'(\alpha^{-(N-1)})$ is formed as shown and this error sequence is selected as the output.

At the k^{th} step this implementation evaluates the σ, A, and σ' polynomials at $\alpha^{-(N-k)}$, checks to see if the k^{th} received symbol is in error, computes the value of the error if there is one, and outputs an estimate of the superchannel bit error sequence corresponding to the k^{th} received symbol.

2.5.6 Hardware Implementation Summary

Table 2.5.1 summarizes the number of chips required to hardware implement the various operations in this concatenated coding system. This table shows that most of this system can be implemented with TTL logic. MOS chips are used only in the unscrambler and in the delay line storage during R-S decoding, where large amounts of storage are required. A total of 17, 4096 x 1 MOS RAM's are used for these storage purposes.

The table shows that the decoder for this concatenated coding system can be implemented with only a little over twice the number of chips as the basic Viterbi decoder. That is, it requires a few more chips than a $K = 9$, $R = 1/3$ Viterbi decoder. However, this concatenated coding system only requires 1.93 and 2.18 dB to achieve bit error probabilities of 10^{-4} and 10^{-7}, respectively, while the $K = 9$, $R = 1/3$ Viterbi decoder system requires about 2.6 and 4.2 dB. To obtain the same performance as this concatenated coding system, a considerably longer and exponentially more complex Viterbi decoder system would be required.

FUNCTION	Number of TTL chips	Number of MOS chips	Total Number of chips
R-S Encoder, Sync Circuit, and Interleaver	31	0	31
Viterbi Encoder	6	0	6
Total Encoder	37	0	37
Viterbi Decoder	150	0	150
Concatenated Code Array Sync	8	0	8
Unscrambler	14	16	30
Storage during R-S Decoding	4	1	5
Syndrome Calculation	27	0	27
Berlekamp Algorithm	42	0	42
Chien Search and Error Evaluation Preparation	31	0	31
Chien Search and Error Evaluation	40	0	40
Total for R-S Decoder without Sync	144	1	145
Additional Chips Required to Convert a Viterbi Decoder to a Concatenated System Decoder	166	17	183
Total Decoder	316	17	333

Table 2.5.1 Summary of the estimated number of chips required to hardware implement a concatenated coding system with a K=8, R=1/3 convolutional inner code and a J=8, E=16 R-S outer code with I=16 levels of interleaving.

3.0 Hybrid Bootstrap Decoding

Performance results for hybrid bootstrap decoding[16,17,18] based on extensive simulations by Norman are presented in two papers (Refs. 14, 15). The design considered in Section 3.2 follows these papers very closely, since alternate schemes have not been adequately tested. We are particularly interested in the rate 1/3, one group, soft-decision decoder without multiple processing, which achieves performance comparable to that of concatenated coding. Performance is reviewed in Section 3.1 and an implementation based on MECL 10,000 logic is presented in Section 3.2. Suggestions and comments on other approaches are contained in Section 3.3. A careful comparison and evaluation of hybrid and concatenated coding is contained in Section 4.

3.1 Performance Results

In hybrid decoding, the principal source of failure is block erasure due to inadequate time to decode. Undetected errors also occur. An undetected output bit error rate of 2.5×10^{-6} near R_{comp} is cited in Ref. 14 for the rate 1/2 code. It is anticipated, however, that with proper choice of parameters and during operation at rates below R_{comp}, that is, with E_b/N_o of 1.5 dB or higher, the undetected error rate will be significantly lower than 10^{-6} and not a significant cause of system degradation.

An erasure occurs whenever the number of computations required to decode a block exceeds the number of computations that can be performed by the decoder during the time allotted for decoding the block. In real-time decoding, this number is approximately equal to the computational speed of the decoder times the time required to transmit one block. Buffering external to the decoder will permit additional time to be devoted to difficult blocks, beyond that required to transmit the block, but this effect is not major unless very large buffers (and delays), or off-line processing, is provided.

Fig.3.1.1 is an extrapolation of the results of Fig. 1 of Ref. 15, the normalized computational distribution for a rate 1/3, 7-track bootstrap decoder. These curves may be used to approximate system performance as follows. A decoder capable of performing D computations per second can perform $L_T = D \times 3000/R$ computations during the time required to transmit a block of 3000 bits at an information bit rate of R bits per second. The normalized total number of computations is obtained by dividing L_T by the number of information bits, yielding

$$\mu = L_T/3000 = D/R.$$

Thus, the normalized total number of computations per block is just the computational speed factor, μ, defined as the ratio of the computational rate of the decoder to the information bit rate. For a decoder capable of 15 megacomputations

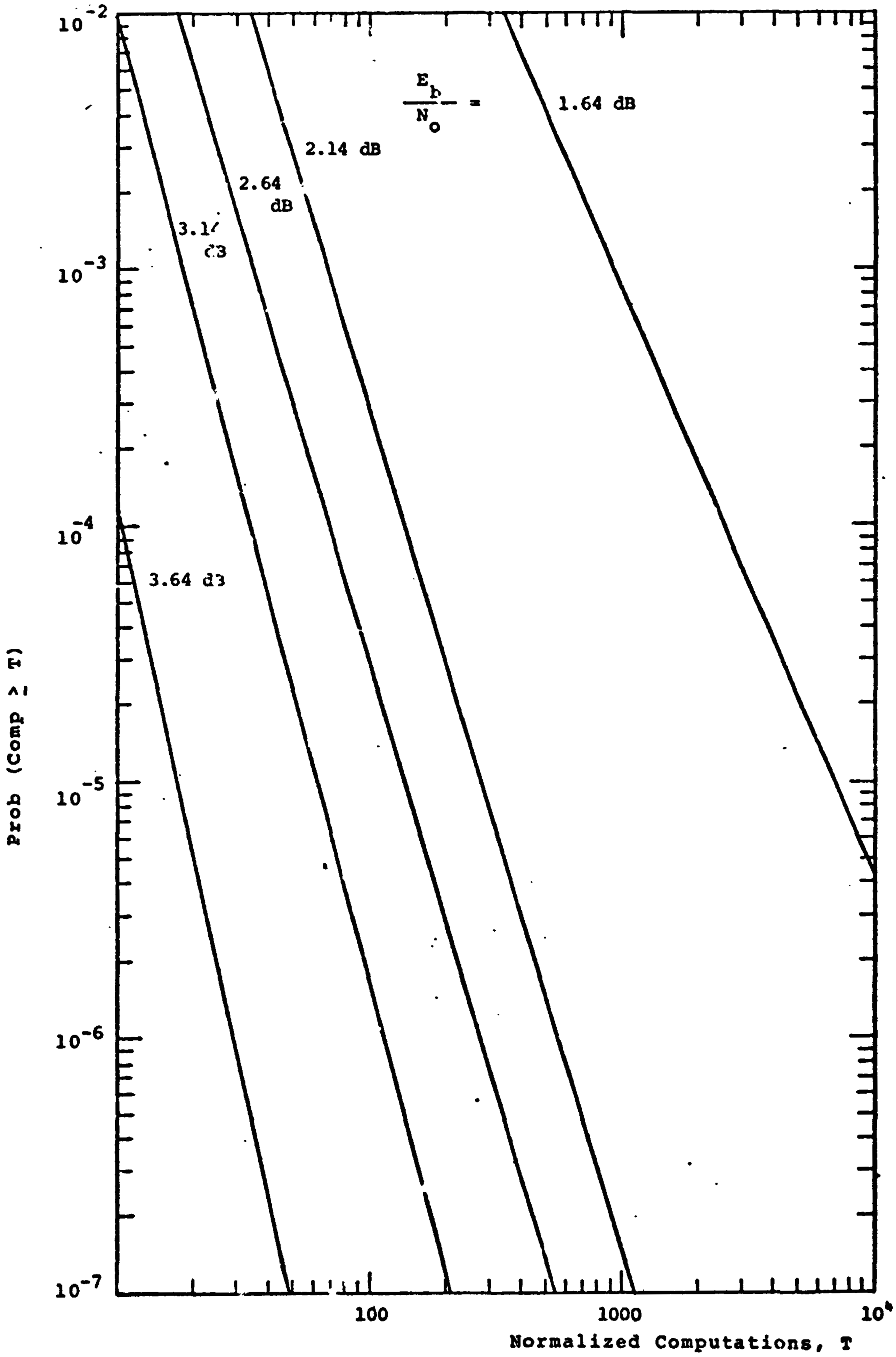


Figure 3.1.1 Extrapolated Distributions, Octal Hybrid Sequential Decoder.

per second (MCPS), μ = 150 at a rate of 100 Kbps and μ = 1500 at a data rate of 10 Kbps. The curves of Fig. 3.1.1 then indicate an erasure rate of 7×10^{-5} at an E_b/N_o of approximately 2.2 dB at 100 Kbps and an erasure rate of 3×10^{-4} at an E_b/N_o of approximately 1.7 dB at 10 Kbps.

Curves of erasure vs. E_b/N_o at data rates of 10 and 100 Kbps assuming a 15 MCPS decoder are presented in Fig. 3.1.2. It should be noted that these curves are based on rather uncertain extrapolation and thus are subject to considerable inaccuracy. The performance of a 16 error correcting concatenated Reed-Solomon Viterbi decoder is also shown in Fig. 3.1.2 as a curve of block error probability vs. E_b/N_o. The hybrid decoder operating at 10 Kbps appears to have a slight performance advantage down to block error or erasure probabilities of 10^{-7}. At lower speed factors, hybrid decoding appears to suffer badly. In particular, at 100 Kbps, hybrid decoding is quite inferior for block erasure probabilities less than 10^{-4}.

The reason for this inferior performance is not clear. Fig. 2 of Ref. 15 shows an unexplained decrease in Pareto slope, α, for the rate 1/3 code as E_b/N_o is increased from 2 to 3 dB. It is this decrease that shows up as inferior hybrid decoding performance at 100 Kbps above 2 dB. Whether this is a basic problem, a quirk in the implementation, or overly ambitious extrapolation remains to be explained.

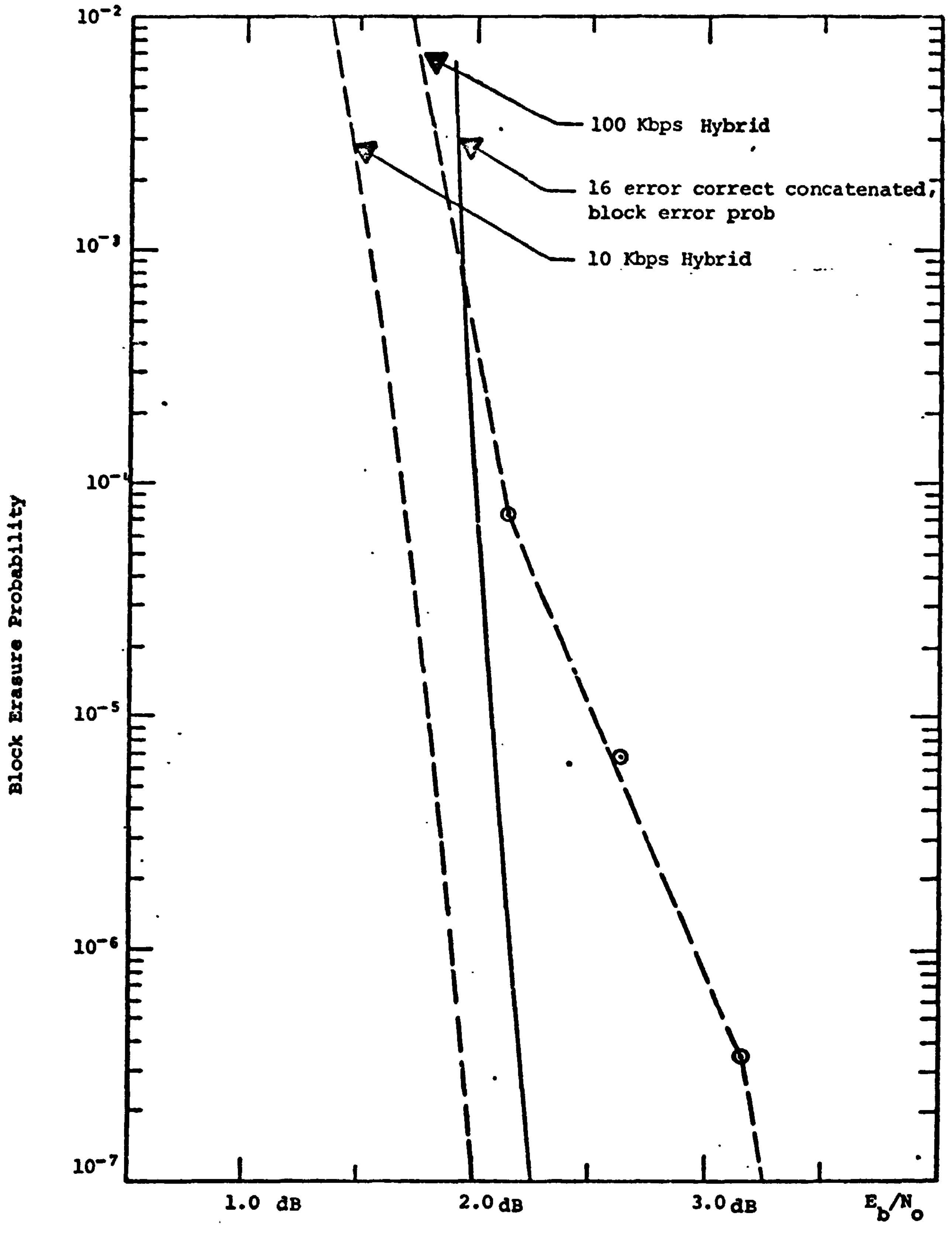


Figure 3.1.2 Erasure Probability for 15 MCPS Hybrid Sequential Decoder, Rate 1/3, Octal Channel

PRECEDING PAGE BLANK NOT FILMED

It is clear that there are significant advantages to high speed factor. The design for a soft decision, rate 1/3 hybrid decoder is discussed in Section 3.2. A faster computation rate does not presently appear to be practical.

3.2 Hybrid Bootstrap Sequential Decoder Implementation

A design of a hybrid sequential decoder, using the algorithm presented in Ref. 14, is described in this section. A block diagram of the decoder is shown in Fig. 3.2.1. When the decoder completes the decoding of a block, the received symbols for the next block are loaded into the decoder memory while the decoded data from the previous block is being read out. Simultaneous with received symbols being loaded, the state track is generated and loaded into the decoder memory.

Three state bits are generated from the received symbols for each of the 512 words in the block. The state bits are computed as follows: the first state bit of word n is equal to the even parity of the sign bits of received symbol one for each of the seven tracks in word n; the second state bit is equal to the even parity of the sign bits for received symbol two, etc. Three more state bits in word n are the binary representation of KLEFT, the number of tracks that have not yet decoded past word n. When the memory is first loaded, KLEFT is set equal to seven in all 512 words. The final bit of the state, referred to as the alternate branch state bit, is particular to the track presently being decoded and is set equal to one on a forward move

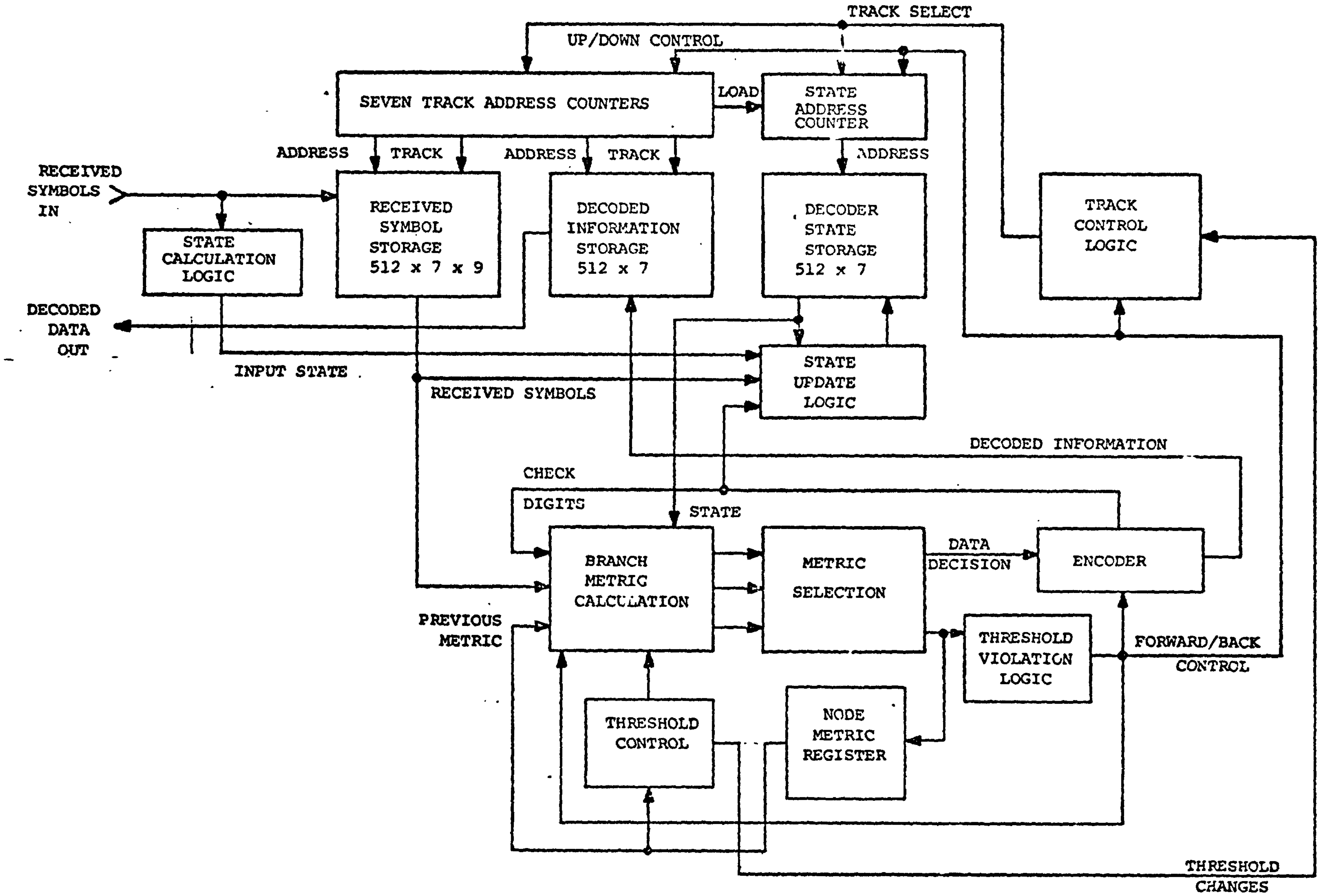


Figure 3.2.1 HYBRID BOOTSTRAP SEQUENTIAL DECODER.

along the best branch from a node, and to 0 on a forward move on the alternate branch.

3.2.1 Decoder Memory Organization

A total memory size of 512 words is required. The total memory is divided into three sections; one for received symbol storage, one for information bit storage, and one for decoder state storage. The received symbol and the information bit sections are divided into seven independent tracks. Each track has its own address counter. The received symbol and information bit storage for a given track share the same address counter. The state memory is addressed by an address counter which is loaded from the track address counter of the track currently being decoded.

The received symbol storage requires a nine-bit word for each track for a total of 512 x 7 x 9 = 32,256 bits of storage. The information bit storage requires only one bit per track for a total of 512 x 7 = 3,584 bits. The state memory has only a single track. A seven-bit word is required for a total of 3,584 bits. Thus, the total storage required is 39,424 bits. The cycle time must be approximately 50-60 ns. It appears that these requirements can best be met with the Fairchild 95410, 256-bit ECL memory. A total of 154 of these devices are required to build this memory.

While the decoder is computing the present node, the memory reads out the received symbols and state bits for the next computation. Since the decoder may move either forward or backward, the received symbols and state bits for both the next node and the previous node must be provided.

3.2.2 Decoding Logic

Since the decoding logic speed determines the overall computation rate, it has been worked out in some detail. A block diagram of the decoding logic is shown in Fig. 3.2.2. The decoding logic has two modes, the look-forward mode and the look-back mode. The decoder is in the look-forward mode if the present node was arrived at by a forward step. Otherwise, the decoder is in the look-back mode.

The node metric, or MT, register contains the cumulative metric minus the cumulative threshold for the present node. A 16-bit register for use with symbol metric values quantized to 12 bits is assumed, based on simulations performed by L. Hofman and summarized in Fig. 3.2.3. The 2 curves encompass a range of choices of metric quantization and of KLEFT. Hofman notes that, by extrapolation, a 16-bit MT register can be expected to overflow about once every 5×10^{33} blocks, whereas a 14-bit register could be expected to overflow every 5×10^{4} blocks when used with 12-bit symbol metrics. The choice of 16 bits thus appears to be reasonable. Symbol metric quantization is discussed in connection with Fig. 3.2.4.

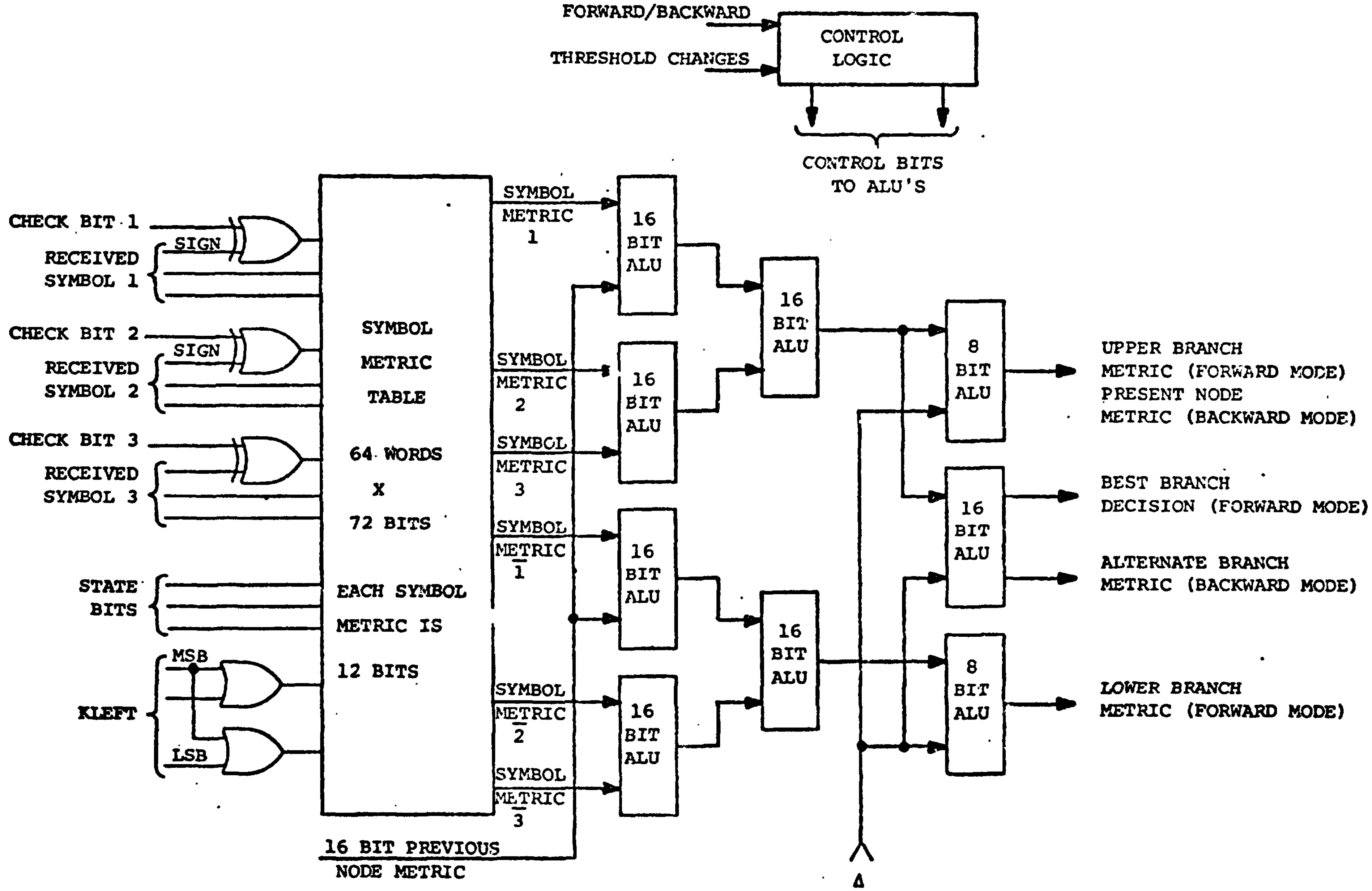


Figure 3.2.2 Hybrid Bootstrap Sequential Decoder Branch.

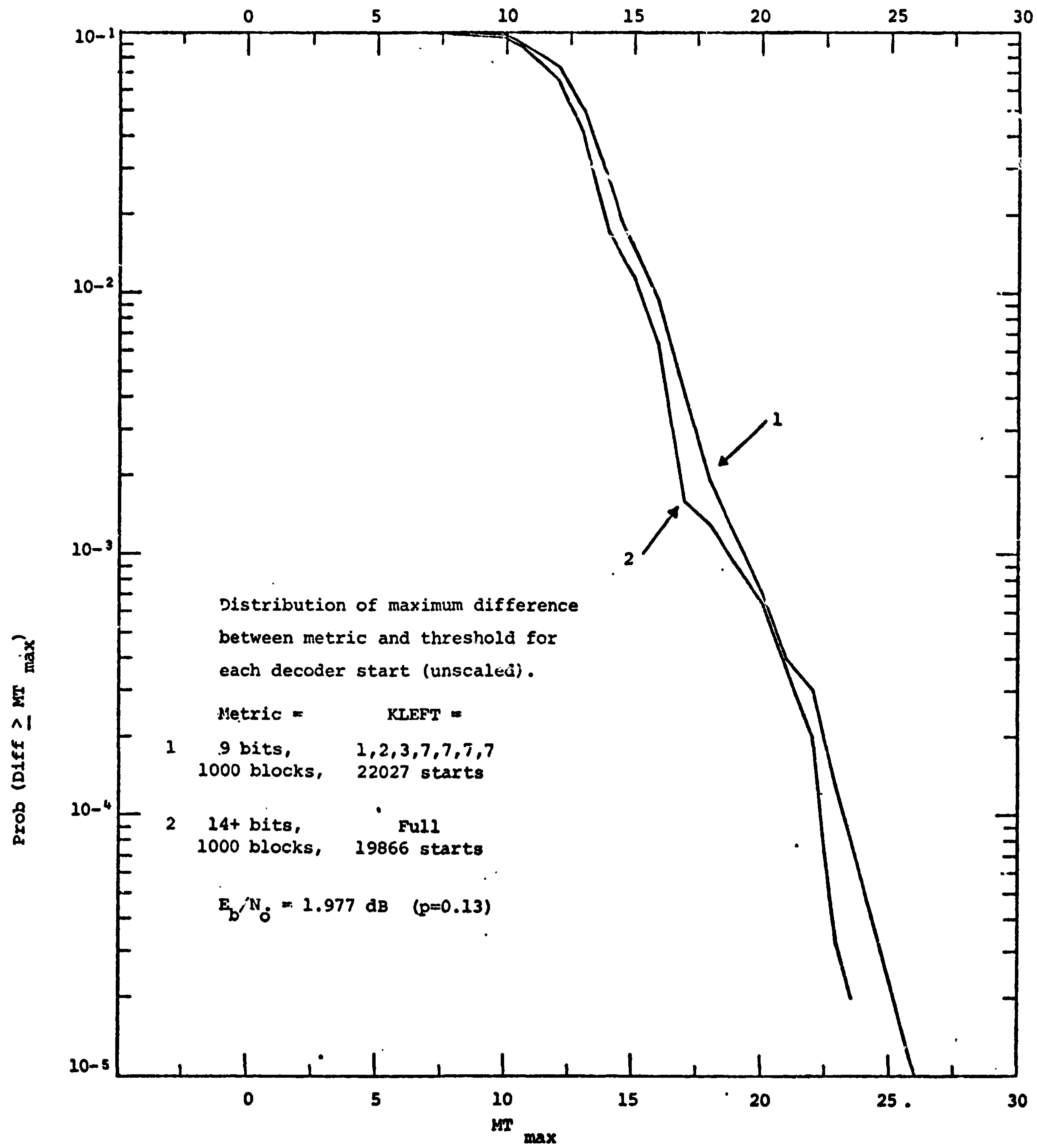


Figure 3.2.3. Complementary Distribution Function for MT for rate 1/2, inner code, 7-track Hybrid Bootstrap Decoder.

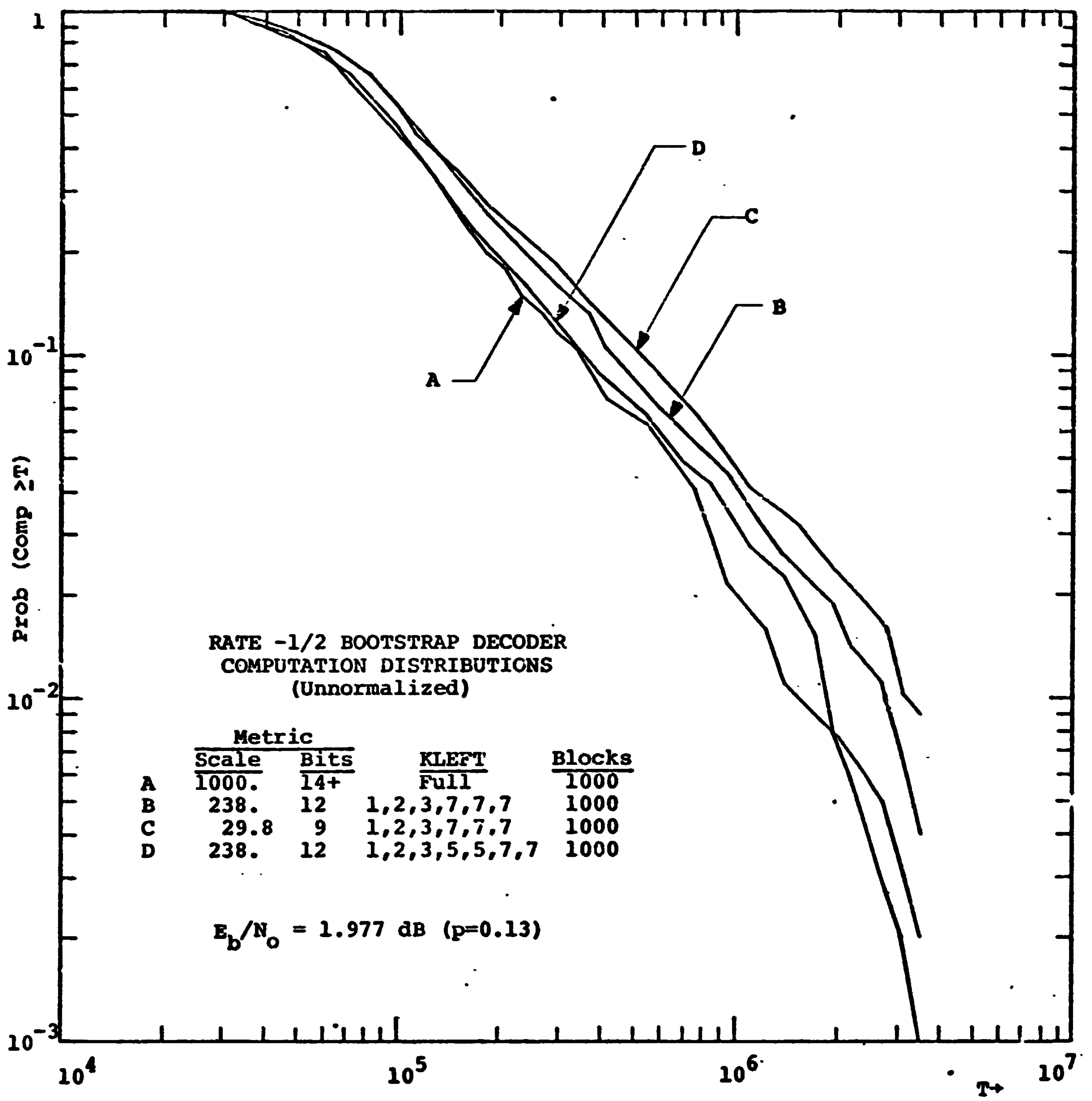


Figure 3.2.4 Sensitivity of Bootstrap Decoder Computations to Quantization of Metrics and of KLEFT.

The MT register is set to zero when starting or resuming the decoding of a track. When in the forward mode, the metric calculator simultaneously computes the metric for the two successor nodes to the present node by adding the three symbol metrics for each branch to MT. The best of the two metrics is tested for threshold violation (negative value of MT). If threshold is violated, the decoder steps back to the previous mode. Otherwise, the decoder steps forward, sets the alternate branch state bit to 1, and tests the best metric for a possible threshold tightening. The metric MT is decreased by Δ if the previous metric was less than Δ and the best new metric is greater than or equal to Δ. The resulting metric is then stored in the metric register and the decoder steps forward on the best branch.

In the back-up mode, the metric of the present branch and the alternate branch is calculated simultaneously. If the present metric is below threshold, then the threshold is loosened by adding Δ to MT and the decoder steps forward on the best branch. If this is not the case, and if the alternate branch available state bit is 1, then the metric of the alternate branch is tested for threshold violation. If the alternate branch metric is above threshold, then the decoder steps forward to the alternate branch, setting the alternate branch available state bit to 0; otherwise, the decoder steps back.

The branch metrics are computed from the symbol metrics and the previous node metric. The only practical way of generating the symbol metrics is by storing the values in three identical look-up tables, one for each symbol. Each look-up table is composed of six 256-bit MECL 10,000 ROM's and is addressed by three bits. These devices (soon to become available) will have access times of about 17 nanoseconds. Each symbol look-up table provides two sets of symbol metrics; one for the upper (0) branch and one for the lower (1) branch. Each symbol metric is stored to 12-bit precision. Initial simulations by Hofman indicate that with appropriate choice of KLEFT quantization to 2 bits, metric table quantization to 12 bits has negligible impact on computational requirements. A more extensive simulation appears to be indicated, however, before parameter choices are frozen. Hofman's results are presented in Fig. 3.2.4. Although obtained for a rate 1/2 code, no differences are anticipated for a rate 1/3 code.

In forward mode, the two branch metrics are formed by summing the symbol metrics with the contents, MT, of the node metric register. These two results are then subtracted from each other to determine the larger of the two. Threshold changes are obtained by adding or subtracting Δ from both the upper and lower branch metrics while they are being compared.

In the backward mode, the present node metric is determined by subtracting the upper branch metric from MT. The alternate node metric is computed by adding the alternate branch metric to the present node metric.

The appropriate metric is selected by a three input multiplexer and stored as the new value of MT in the node metric register. The decision which determines the best metric also determines the information bit. The information bits are shifted into an encoder which determines the check bits for the next computation. After the information bits shift through the encoder, they are stored in the appropriate track of the information bit memory.

As the decoder moves forward, the state bits are updated. Each check bit from the encoder is exclusive-OR'd with the sign bit of the corresponding received symbol. The result is exclusive-OR'd with the corresponding state parity bit and stored as the new state parity bit. At the same time, the quantity, KLEFT, is decreased by one. When backing up, KLEFT is increased by one and the state parity bits are changed back to their original condition. The alternate branch state bit is set to 1 or 0, depending on whether the forward move is along the best or worst branch, respectively.

3.2.3 Track Control Logic

The function of the track control logic is to monitor the performance of the decoder and to switch to another track when the decoder bogs down on the present track. The decoder's progress on the present track is monitored by a counter which is incremented when the decoder threshold is loosened. The counter is reset to zero when the decoder tightens threshold. The number in this counter is continuously compared with a stopping threshold, DSTOP. If the threshold is violated, then the track control logic switches the decoder to the next unfinished track.

The decoder's penetration is also monitored by an up/down counter which is zeroed when decoding switches to a new track. If, when the DSTOP threshold is violated, the decoder has penetrated far enough that progress has been made, a register, KROUND, is reset. Otherwise, KROUND is incremented by one.

If KROUND becomes equal to the number of unfinished tracks, then progress is no longer being made by any of the unfinished tracks. In this case, the unfinished tracks are restarted at the beginning of the block and the stopping threshold, DSTOP, is loosened.

The stopping threshold, DSTOP, is a function of KLEFT and DI, the number of times the unfinished tracks have been initialized. The quantity, KLEFT, is stored in the state memory. The quantity, DI, is the contents of a 2 bit counter, initially zero, which is incremented whenever all unfinished tracks become stalled, as determined from KROUND equaling KLEFT. The DI counter is reset whenever a new track is finished. The stopping thresholds are stored in 32 words of a single MECL 10,000 ROM addressed by KLEFT (3 bits) and DI (2 bits).

When the stopping threshold, DSTOP, is violated, then the track control logic stops the decoding of the present track and begins the decoding of a new track. If KROUND equals KLEFT, all unfinished tracks are stalled and all uncoded tracks are reinitialized. The simulations of Ref. 14 and 15 assume restarting at the track origins. However, some time and probably computations would be saved if reinitialization were achieved by starting at a point between the origin and the present node, that is, by backing up a fixed distance after stalling.

When switching to a new track, only those nodes which lie ten nodes or more behind the present node are considered to be "definitely" decoded. Since the state has been updated to the present node, the decoder is backed up ten nodes, thusly restoring the state bits of non-definitely decoded nodes to their previous values. To provide correct information for the next restart, the decoder is then forced forward 24 nodes, with all decoding operations suspended, thus storing the encoder contents in the information bit memory.

Track change is then accomplished by incrementing the 3 bit track pointer counter which selects the active track. The track address counter of the new active track is checked to see if this track is completely decoded. If so, the track pointer counter is incremented until an unfinished track is found.

Decoding of this track is started by first loading the encoder by forcing the decoder to back up 24 nodes. The metric register and the progress counter are then reset to zero. The present node then forms a "pseudo origin" for the subsequent decoding operations. This completes the switching process and the decoder is allowed to proceed until the stopping threshold is violated again, or until the track is completely decoded.

In the event that all unfinished tracks are stalled, then they must be restarted at the beginning of the block or at intermediate points. But first the state must be cleared of the effects of the unfinished decoders. This is accomplished by loading each unfinished track into the decoder and forcing it to back up a fixed number of nodes or to the beginning of the track. Decoding then resumes but with a looser stopping threshold.

3.2.4 Parts Count Estimation

The part count necessary to implement the decoder has been estimated. The estimate is based on the use of presently available ECL 10,000 and 9,500 series logic circuits. The MECL 10139 ROM or equivalent has been assumed to be available in the near future. The parts count has been broken down as follows:

Memory and Associated Registers	170
Metric Calculation	50
Metric Testing and Selection	30
State Calculation & Update Logic	15
Track Control Logic	30
Encoder	25
Memory Address Registers	30
Miscellaneous	50
External Buffer	50
TOTAL	450

Table 3.1.1 Approximate I.C. Requirements for Hybrid Bootstrap Sequential Decoding.

This number of circuits can be packaged on 5-6 circuit boards approximately 8x8 inches in size. Prime power requirements are approximately 400 watts, assuming 50% power supply efficiency.

3.2.5 Decoder Computation Rate

The part of the decoder that determines the maximum computation rate is the branch metric calculation and selection circuitry shown in Figure 3.2.2. Note that the entire branch metric computation is done in one computational cycle. The total delay through this circuitry is approximately 60 nanoseconds including set-up and propagation delays of the flip-flop registers involved. This is the basis for the 15 Megacomputations per second decoding speed forecast.

It is possible to speed up the process by the use of pipeline techniques; i.e., by doing part of the metric calculation on the previous computational cycle. The difficulty here is that whatever portion of the hardware operates on the previous cycle must compute branch metrics for three times as many nodes. This is because the present computational cycle may step back or step forward to two different nodes and symbol metrics have to be provided for all three possibilities.

The use of MECL III in the symbol metric summers was considered briefly and rejected in favor of the ECL 10181 arithmetic logic unit. It was found that only a small improvement could be made in propagation delay at greatly increased chip count and cost. Actually, the increased size of the resulting circuit board layout would probably cancel the smaller propagation delay because of increased wire length.

3.3 Other Bootstrap Decoding Techniques

The design of Section 3.2 is based on the best understood of the bootstrap sequential decoding techniques. The basic hybrid bootstrap decoding algorithm is well suited for hardware implementations, but initial simulation results do not indicate any clear performance improvement over concatenated convolutional - RS decoding which is somewhat simpler to implement.

3.3.1 Multiple Processors

Hybrid bootstrap decoding performance could be improved if the speed factor of the sequential decoder could be effectively increased by factors greater than 2 without significant cost increments. One approach with potential promise is utilization of multiple processors. Initial simulations, discussed by Hofman and Odenwalder [15], demonstrated a reduction in performance. The problem appears to reside in the communication problem among the processors and, in particular, in techniques for revising state information and recognizing definitely decoded sections without introducing errors. Each sequential decoder must be able to accept changes in branch metric assignments without complete initialization, without looking, and without significant computational increases. Further work is indicated. †

---

† Section 3.3.2 was authored by Dr. F. Jelinek, a consultant to LINKABIT on this study. He considers application of bootstrap techniques to Viterbi (trellis) decoding with long constraint length codes.

3.3.2 Bootstrap Trellis Decoding

3.3.2.1 Description of Rudimentary Decoder

Let K be the constraint length of a convolutional code, and let the constraint length of the corresponding truncated trellis decoder be $\mu<K$ (i.e., the truncated decoder has $2^{\mu-1}$ states per level). We will assume that K is so large that the probability of error for maximum likelihood decoding at the signal-to-noise ratio (SNR) used is negligible compared to the probability of error resulting from the scheme described below.

The rudimentary Bootstrap Trellis decoding algorithm is as follows:

1. m-1 streams of binary data are encoded using the K constraint length code, and an m^{th} stream is created using mod 2 position-by-position addition of the M-1 streams.

2. The m streams are transmitted through the channel, and the receiver creates an appropriate state stream as in Bootstrap Sequential Decoding.

3. A μ-truncated trellis decoder is used to decode the first stream, with metrics based on the corresponding received and state stream digits. To each depth of the N-branch codeword there correspond 2^{m-1} likelihoods, the maximum of these at depth n being denoted by L_n.

Let

$$L_n^M = \max_{1 \le i \le n} L_i$$

so L_n^M is a monotone increasing function of $n \varepsilon (1,\ldots,N)$. If $L_n^M - L_n < T$ for all n, the decoder accepts the decoded first stream information sequence, otherwise it rejects it (in fact, it will stop decoding after smallest depth n is searched for which $L_n^M - L_n \le T$).

4. If the 1^{st} stream was accepted, it is replaced by the estimated transmitted stream, the state stream is accordingly recalculated, and the decoder proceeds to decode the 2^{nd} stream as in step 3, using a metric table appropriate to m-1 undecoded streams.

5. If the 1^{st} stream was rejected, 2^{nd} stream decoding proceeds exactly as in (3) with no change to either metric or state stream.

6. Steps 3 through 5 establishes a pattern that is adhered to in general: after every acceptance, the state stream and metrics are recalculated and decoding of the "round robin" next stream begins.

7. Decoding terminates in either of two ways:

   a) SUCCESS: all m streams get finally accepted.

   b) FAILURE: when ℓ streams, ($\ell \le m$), remain undecoded, ℓ successive attempts at stream decoding end with rejection.

3.3.2.2 Analytical Performance Estimates

Using simple arguments, analogous to the ones appearing in the Bootstrap Sequential Decoding paper,[16] it is possible to obtain bounds on the probability of DECODING FAILURE, P(F).

Let E_k (R) be the probability of undetected error exponent corresponding to maximum likelihood decoding of the first of k streams that utilize the received as well as state stream digits when the convolutional rate is R (the net rate taking into account the parity stream degradation is $\frac{m-1}{m}$ R). Then

$$P(F) = \begin{cases} \leq \max\limits_{2\leq k\leq m} \min \left\{ \left[NA_\infty 2^{-\mu E_\infty (R)} \right]^k, \; NA_k 2^{-\mu E_k (R)} \right\} \\ \geq \max \max\limits_{3\leq k\leq m} \left\{ \left[NA_k 2^{-\mu E_k (R)} \right]^k, \; NA_2 2^{-\mu E_2 (R)} \right\} \end{cases} \tag{3.3.1}$$

where A_ℓ is a monotonically increasing function of the number of undecoded streams ℓ that depends on the rate R but varies negligibly with μ. From (3.3.1) it follows that

$$E_{LB} (R) \leq \lim_{\mu\to\infty} - \frac{1}{\mu} \log P(F) \leq E_{UB} (R) \tag{3.3.2}$$

where estimates of both exponents are readily available. In fact, let us assume that the combined expurgated and random coding bound exponents are the true exponents.

Then

$$E_k(R) = \begin{cases} \sigma & \text{if the solution of } R = \frac{1}{\sigma} E_k^o(\sigma) \text{ is } \sigma \le 1 \\ \beta & \text{if the solution of } R = \frac{1}{\beta} E_k^x(\beta) \text{ is } \beta > 1 \\ 1 & \text{otherwise} \end{cases} \qquad (3.3.3)$$

where, for the binary input, 2b-ary output symmetrical channel, and binary state stream,

$$E_k^o(\sigma) = \sigma - \log \sum_{v=1}^{b} \left(\left\{ \left[w(0,v|0) q_{k-1}(0) \right]^{\frac{1}{1+\sigma}} + \left[w(1,v|0) q_{k-1}(1) \right]^{\frac{1}{1+\sigma}} \right\}^{1+\sigma} + \right.$$

$$\left. + \left\{ \left[w(0,v|0) q_{k-1}(1) \right]^{\frac{1}{1+\sigma}} + \left[w(1,v|0)\, q_{k-1}(0) \right]^{\frac{1}{1+\sigma}} \right\}^{1+\sigma} \right)$$

Above, the channel outputs are pairs (u,v), $u \in (0,1)$, $v \in \{1,\ldots,b\}$ and the inputs are $x \in (0,1)$. The transmission probability, $w(u,v|x)$ is symmetric: $w(u,v|x) = w(u \oplus 1, v|x \oplus 1)$

Furthermore,

$$q_{k-1}(0) = \frac{1+(1-2p)^{k-1}}{2},$$

$$q_{k-1}(1) = \frac{1-(1-2p)^{k-1}}{2}$$

$$p = \sum_{v=1}^{b} w(1,v|0)$$

Finally,

$$E_k^x(\sigma) = \sigma - \sigma \log\left[1 + \sum_{v=1}^{b} \sqrt{w(o,\ v/o)w(1,v/o)q_{k-1}(0)q_{k-1}(1)}\right]$$

Clearly, for (3.3.1) and (3.3.2)

$$E_{LB}(R) = \min_{2 \le k \le m} \max \left\{kE_\infty(R),\ E_k(R)\right\}$$

$$E_{UB}(R) = \min\left\{\min_{3 \le k \le n}\left\{k\ E_k(R)\right\},\ E_2(R)\right\}$$

To obtain parametric relations between $E_{LB}(R)$ and $E_{UB}(R)$ and R, we may proceed as follows. Define

$$E_k(\alpha) = \begin{cases} E_k^o(\alpha) & \alpha \le 1 \\ E_k^x(\alpha) & \alpha > 1 \end{cases}$$

Then $E_{LB}(R) = \alpha$ for

$$R = \max\left\{\frac{1}{\alpha} E_m(\alpha),\ \min\left\{\frac{1}{\alpha} E_{k^+-1}(\alpha) \quad \frac{k^+}{\alpha} E_\infty\left(\frac{\alpha}{k^+}\right)\right\}\right\}$$

Where

$$k^{+} = \min \left\{ k : k \geq 2, \ \frac{1}{\alpha} E_k(\alpha) \geq \frac{k}{\alpha} E_\infty\left(\frac{\alpha}{k}\right) \right\}$$

Finally,

E_{UB} ' $= \alpha$ for

$$R = \min \left\{ \frac{1}{\alpha} E_2(\alpha), \ \frac{k^*}{\alpha} E_{k^*}\left(\frac{\alpha}{k^*}\right) \right\}$$

where

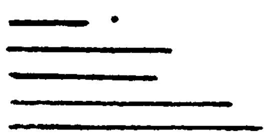
$$k^* = \min \left\{ m, \ \min \left\{ k : k \geq 3, \ \frac{k}{\alpha} E_k \left(\frac{\alpha}{k}\right) \leq \frac{k+1}{\alpha} E_{k+1} \left(\frac{\alpha}{k+1}\right) \right\} \right\}$$

3.3.2.3 Refinements of the Decoding Algorithm

1. It is not necessary to reject entire stream when threshold violated (i.e. at some $\ell, L_\ell^M - L_\ell \geq T$). Let $k<\ell$ be such that $L_\ell^M = L_k$ and let J be an optimized integer. Then when threshold violation occurs at depth ℓ, all decoded bits up to the $K-J^{th}$ one are accepted, the corresponding state stream digits are recalculated, and on subsequent attempts the appropriate metrics are used. The next decoding of that stream then starts at position k-J, rather than at position 1.

This pull-up strategy will occasionally introduce errors into the accepted stream sections, so an error-cleanup method must also be agreed on. There is furthermore the problem that J should probably increase with μ, but this may not be serious in the "reasonable" range $\mu \leq 10$.

2. If the pull-up strategy of (1) is used, then at the end of the first m attempts, the length of the definitely decoded sections will have a monotone increasing tendency, e.g.:



For this reason it might be useful to modify the round robin strategy by next decoding backwards starting with the last not fully accepted stream and continuing with the next-to-last stream, etc. After recoding the first stream in this manner, decoding would start again in the forward direction, etc.

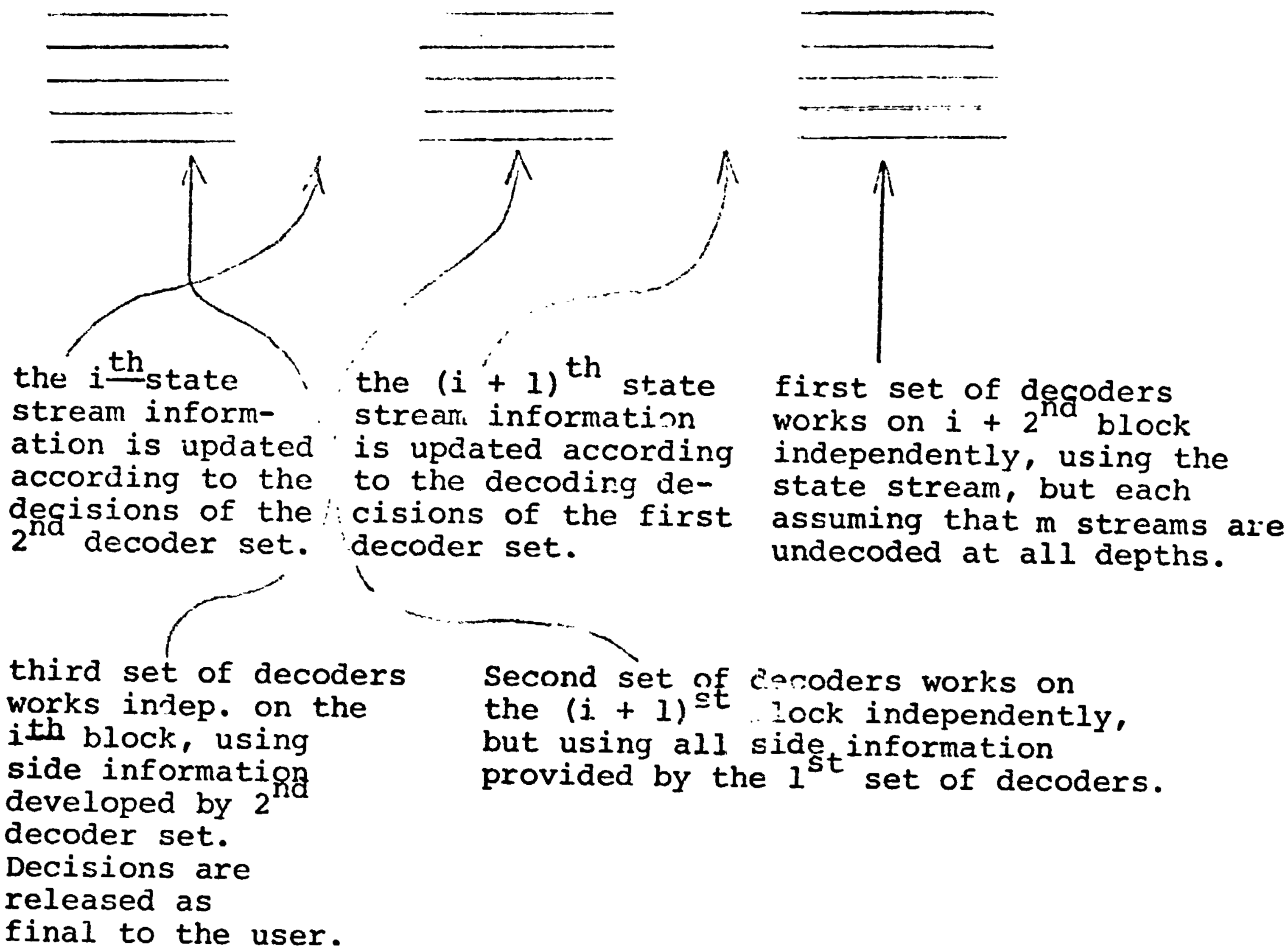
Unlike a true maximum likelihood decoder, a truncated decoder is not symmetrical in both directions. Therefore, backward decoding might avoid some errors coded in the forward direction and vice versa. The code should be picked so it is strong in both directions.

3. More complex algebraic codes ought to be considered, such as the three group code.

4. A decoding failure does not mean that the entire block must be thrown away. In fact, in the definately decoded sections there will probably be no errors whatever. When FAILURE takes place, one should probably go one more round, ignoring the threshold stopping rule and decoding each stream to its end, simply accepting the admittedly unreliable decoder decisions. With a systematic or quick-lookin code, one might reconstruct the unreliable positions simply from the uncoded received information bits.

5. The final stopping rule that would minimize the probability of error in the pull up mode (1) would be that failure results when further decoding results in no enlargement of definitely accepted stream sections.

Alternately, it might be desirable to fix the number of allowed decoding attempts on each track, perform these, and accept the last decisions regardless of whether additional progress was being made or not. This approval might be particularly useful if several truncated decoders working in parallel would be available. As one possibility, say 3 sets of m decoders would work continuously on 3 successive received blocks as follows:


the i^{th} state
stream inform-
ation is updated
according to the
decisions of the
2^{nd} decoder set.
the $(i + 1)^{th}$ state
stream information
is updated according
to the decoding de-
cisions of the first
decoder set.
first set of decoders
works on $i + 2^{nd}$ block
independently, using the
state stream, but each
assuming that m streams are
undecoded at all depths.
third set of decoders
works indep. on the
i^{th} block, using
side information
developed by 2^{nd}
decoder set.
Decisions are
released as
final to the user.
Second set of decoders works on
the $(i + 1)^{st}$ block independently,
but using all side information
provided by the 1^{st} set of decoders.

6. One way to avoid rejection is to increase the truncated constraint length μ. Thus, for example, in the rudimentary decoding algorithm of section I, one would have a sequence $\mu_1<\mu_2<\ldots<\mu_\ell$ of constraint lengths.

Starting with constraint length μ_1, decoding of a block would be attempted until either a SUCCESS or a FAILURE was declared. In the latter case, decoding would begin again based on constraint length μ_2 and would continue until either a new stream was accepted, or another FAILURE resulted. In the former case, decoders would revert to constraint length μ_1; in the latter case constraint length would be increased to μ_3, etc. FAILURE with constraint length μ_1, would be final.

This game could be played in a variety of ways. Another possibility is to have a sequence of constraint lengths $\mu_2\leq\mu_3\leq\ldots\leq\mu_m$ (m is the number of streams). Constraint length μ_m is used until one stream is accepted or FAILURE is declared. In the former case, constraint length μ_{m-1} will be used until another stream is accepted or FAILURE is declared.

7. It is not clear that the pull-up strategy of (1) is in the non-asymptiotic case more desirable than the rudimentary one. Indeed, it may be much simpler to shorten the stream length N and use the latter strategy.

8. The final possibility is to bootstrap on a concatenated code. To stay simple, suppose an R-S code is used over GF(8) that is single error correcting (not realistic, of course). The algebraic rate then is $\frac{5}{7}$.

Form 5 information streams and add 2 parity check streams using the R-S relation. Next encode each of the streams by use of rate 1/2 convolutional code that has eight 6-bit branches leaving each node. The matrix of the convolutional code will have the partitioned form (K = 6 in this example):

$$\begin{bmatrix} G0 & 0 & 0 & 0 \\ G1 & 0 & 0 & 0 \\ G2 & G0 & 0 & 0 \\ G3 & G1 & 0 & 0 \\ G4 & G2 & G0 & 0 \\ G5 & G3 & G1 & 0 \\ 0 & G4 & G2 & G0 \\ 0 & G5 & G3 & G1 \\ & & G4 & G2 \\ & & G5 & G3 \\ & & & G4 \\ & & & G5 \end{bmatrix}$$

Here G_i are 3 x 3 binary metrics that are <u>restricted</u> to have the forms

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\text{or } \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}^n \quad , \; n= 1,2,\ldots, 7$$

In this way G_i's are representations of the powers of the primitive element over GF(8). It can be shown that the restriction does not damage the error correcting capabilities of the code, at least not in the limit of large constraint lengths, $K>>1$.

If a convolutional code of the indicated form is used, then it can easily be shown that the corresponding branch triplets of the 7 streams will have the R-S relationship. In fact, if we number the digits as indicated,

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21

they will satisfy the parity check Matrix

$$H = \begin{bmatrix} 110 & 010 & 010 & 100 & 110 & 100 & 000 \\ 111 & 101 & 101 & 010 & 111 & 010 & 000 \\ 010 & 011 & 011 & 001 & 010 & 001 & 000 \\ 000 & 110 & 010 & 010 & 100 & 110 & 100 \\ 000 & 111 & 101 & 101 & 010 & 111 & 010 \\ 000 & 010 & 011 & 011 & 001 & 010 & 001 \end{bmatrix}$$

Consequently, sets of these digits will be algebraically related, and thus bootstrapping will be possible. If the bootstrapping is to be simple, individual bits in the triplets should get <u>independent</u> metrics. This is not rigorously possible, but from a practical point of view it may be enough to provide three separate parity check equations, each involving only one digit of the triplet. For digits 1,2,3 and parity checks are

100	001	001	101	100	101	000
010	100	110	100	000	110	010
001	001	101	100	101	000	100

It would seem possible to keep state information for each digit according to its parity block set and then perform ordinary single parity bootstrapping. After the decoding of all streams is completed, the R-S code can be used to correct remaining errors in the estimated information streams. Of course, the H-Matrix contains the possibilities for 3-group codes, and even more complicated ones, all of which might be worth investigating.

4.0 Conclusions and Recommendations

Tables 1.0.1, 2.5.1, and 3.1.1 summarize the complexity of the concatenated and hybrid coding systems studied. It appears that the concatenated system is more cost effective for approximately the same performance as the hybrid systems. Its only drawback lies in the interleaving requirements which increase both the decoding delay and the gaps of non-data (mostly parity-check bits) by over an order of magnitude relative to the hybrid system. On the other hand, these probably are not a detriment for time-division multiplexed users, and for systems where this is a problem, staggered interleaving will reduce the gap lengths to those of the hybrid system, possibly at a small cost in decoder complexity.

It is therefore our conclusion that a concatenated coding system utilizing a rate 1/3, constraint length-8, inner convolutional coder-Viterbi decoder, a 2048-bit Reed-Solomon outer block coder-decoder, and 16 words of interleaving, operating at 100 Kbps data rates is implementable in its entirety by a system employing approximately 333 TTL integrated circuits. The coding gain at $P_b = 10^{-7}$ for this system is over 9dB. Hybrid bootstrap sequential decoding would require on the order of 50% more integrated circuits of the ECL-MSI (such as MECL 10,000) logic family. Furthermore, the performance of the latter would vary with data rate, being slightly superior (0.2dB) at 10 Kbps bit and somewhat inferior (1.0 dB) at 100 Kbps.

It should also be emphasized that the simulations of the hybr otstrap system are not completely conclusive, and hence the technical risk is much greater in this system. Among its principal weaknesses are its sensitivity to AGC inaccuracy and phase tracking errors, which has been shown (Ref. 8) to be considerable even for ordinary sequential decoding. In concatenated decoding, on the other hand, these channel inaccuracies produce a moderate known degradation (Ref. 8) on the inner Viterbi decoder, which are easily shown to reflect directly and in almost the same amount on the overall coding scheme.

Otherwise the performance of the two seemingly radically different approaches are remarkably similar. The basic reaso in retrospect, is that, as information theory establishes, highly efficient communication over a Gaussian channel requires extremely long block lengths. The hybrid and concatenated systems considered here utilize about the same "superblock" length - 2 to 3 Kbits - with highly efficient convolutional and block codes - hence the similar performance.

One final advantage of implementing a concatenated scheme is the essentially individual and self-justifying nature of each portion of the system. Viterbi decoders at these data rates exist already (albeit only for the less powerful K=7, R=1/2 code and not for the K=8, R=1/3) and could be inserted without procurement delay as the inner decoders.

The outer Reed-Solomon coder-decoder could be easily justified as a worthwhile development in its own right, since such a powerful (255 characters over GF(2^8) with 16- error-correction capability) decoder has never been implemented in hardware. Finally, even the relatively straightforward interleaver could be justified by itself as a means of breaking up burst errors in convolutional decoding. Thus, such a development would produce multi-purpose components as well as an integrated system which might once and for all conclude the quest for the ultimate coding system for space communications.

Appendix A

This appendix contains a Fortran and an assembly language version of a partial R-S decoder program. Normally the inputs to this program are:

1. J = the number of bits per R-S symbol.
2. E = the designed number of correctable errors.
3. The coefficients of the J-degree primitive polynomial of a field element which generates the field.
4. The 2E syndromes represented as integers.

However, to avoid computing the syndromes by hand, a few additional statements were added at the beginning of the program to enable the computer to determine these quantities from the error locations and error values. Of course, in a real system the syndromes would be computed from the received word. But the method used here is more convenient in checking and timing the various steps in the decoding operation. The program outputs are the error locations and the error values.

The Fortran version of this program contains numerous comment cards describing the various steps in the program. The assembly language version follows the same format as the Fortran version. In addition to the basic IBM 1130 assembly language instructions, a few instructions unique to the LINKABIT system have been used. A brief description of these instructions is given in Table A.1.

Following Table A.1 is a listing of first the Fortran and then the assembly language version of this program.

Mnemonic Code	Instruction
CAR	Copy accumulator to register.
CRA	Copy register to accumulator.
XIR	Interchange index 1 with register.
X2R	Interchange index 2 with register.
X3R	Interchange index 3 with register.
EORR	Exclusive-OR register to accumulator.
SBR	Subtract register from accumulator.
IN	Increase register by one.
DER	Decrease register by one and increase the instruction address register by one on sign change.
DERS	Decrease register by one and increase the instruction address register by one unless there is a sign change.
ICL1	Increase register 1 by one and branch unless the result equals the contents of register 4.
ICL2	Increase register 2 by one and branch unless the result equals the contents of register 4.

Table A.1 LINKABIT Supplement to the IBM 1130 Assembly Language Instruction Set.

```
      INTEGER EOR
      DIMENSION IG(8),IF(7),L(255),IA(255),IS(33),
     1IR1(17),IR2(16),IR3(16),LOC(16),IVALU(16),
     2ISIG(16),LSIG(16),IAA(16),LAA(16),ISIGP(9),LSIGP(9)
     3,INPV(16),INPL(16)
C  INPL = ERROR LOCATIONS
C  INPV = ERROR VALUES
C  IF = COEF. OF MINIMAL POLY. EXCLUDING END COEF.
C  K0 = NO. OF BITS PER SYMBOL
C  NERRO = NO. OF CORRECTABLE ERRORS
C  NEACT = NO. OF ERRORS
      DATA INPL/2,5,7,14,20,30,40,42,101,111,120,121,122,200,203,211/
      DATA INPV/1,2,6,7,101,101,77,51,6,111,201,250,91,92,13,13/
      DATA IF/0,1,1,1,0,0,0/
      K0=8
      NERRO=16
      NEACT=16
      LIM9=2*NERRO
      LIM1=2**K0-1
      LIM2=NERRO-1
      LM1=2*NERRO-1
      INC1=2*NERRO+1
      IS(INC1)=0
      LIM3=K0-1
      LIM5=LIM1-2*NERRO
      LIM8=NERRO+1
      IR1(LIM8)=1
C  SET UP LOG AND ANTI-LOG TABLES
      IG(1)=1
      DO 60 I=2,K0
      IG(I)=0
   60 CONTINUE
      L(1)=0
      IA(1)=1
      LXP1=1
  100 IGL=IG(K0)
      IAX=0
      DO 200 I=1,LIM3
      IND1=K0-I+1
      IND2=K0-I
      IND3=IGL*IF(IND2)
      IG(IND1)=EOR(IG(IND2),IND3)
```

```
      IF(IG(IND1)) 200,200,150
  150 IAX=IAX+2**IND2
  200 CONTINUE
      IAX=IAX-IGL
      IG(1)=IC.
      L(IAX)=LXP1
      LXP1=LXP1+1
      IA(LXP1)=IAX
      IF (LXP1-LIM1) 100,300,300
C   COMPUTE SYNDROMES
  300 DO 304 I=1,LIM9
      IS(I)=0
      DO 304 J=1,NEACT
      IND1=INFV(J)
      LX=L(IND1)+I*(LIM1-INPL(J))
      LX=LX-LIM1*(LX/LIM1)+1
      IS(I)=EOR(IS(I),IA(LX))
  304 CONTINUE
C   SET INITIAL CONDITIONS FOR BERLEKAMP ALG.
      IR1(NERRO)=0
      IR2(NERRO)=0
      IR3(NERRO)=1
      N=0
      IN=0
      IGSTP=1
      DO 310 I=1,LIM2
      IR1(I)=0
      IR2(I)=0
      IR3(I)=0
  310 CONTINUE
C   COMPUTE IDN
 1000 IND1=N+1
      IDN=IS(IND1)
      DO 2000 I=1,NERRO
      IF(IR1(I)) 1100,2000,1100
 1100 IF(IR2(I)) 1200,2000,1200
 1200 IND1=IR1(I)
      IND2=IR2(I)
      LX=L(IND1)+L(IND2)
      IND1=LX-LIM1
```

```
      IF(IND1) 1400,1300,1300
 1300 LX=IND1
 1400 IND1=LX+1
      IDN=EOR(IDN,IA(IND1))
 2000 CONTINUE
C  TEST IF IDN = 0
      IF(IDN) 3000,2010,3000
C  SHIFT R3 RIGHT BLOCK
 2010 DO 2100 I=1,LIM2
      IND1=I+1
      IR3(I)=IR3(IND1)
 2100 CONTINUE
      IR3(NERRO)=0
C  TEST IF N = LM1 TO DETERMINE IF ALG. DONE
 2200 IF(N-LM1) 2300,5000,2300
C  PREPARE FOR ANOTHER CYCLE THRU ALG.
 2300 DO 2400 I=1,LIM2
      IND1=I+1
      IR2(I)=IR2(IND1)
 2400 CONTINUE
      N=N+1
      IR2(NERRO)=IS(N)
      GO TO 1000
C  TEST IF N LESS THAN 2*LN
 3000 IF(N-2*LN) 3100,3500,3500
C  N LESS THAN 2*LN BLOCK
 3100 LD=L(IDN)-L(IDSTN)
      IF(LD) 3300,3310,3310
 3300 LD=LD+LIM1
 3310 DO 3450 I=1,NERRO
      IF(IR3(I)) 3320,3450,3320
 3320 IND1=IR3(I)
      LX=LD+L(IND1)
      IND1=LX-LIM1
      IF(IND1) 3340,3330,3330
 3330 LX=IND1
 3340 IND1=LX+1
      IR1(I)=EOR(IR1(I),IA(IND1))
 3450 CONTINUE
```

```
      GO TO 2010
C   N GREATER THAN OR EQUAL TO 2*LN BLOCK
 3500 LC=L(IDN)-L(IDSTA)
      IF(LC) 3530,3540,3540
 3530 LC=LC+LIM1
 3540 DO 3700 I=1,NERRO
      IND1=IR3(I)
      IF(IND1) 3550,3620,3550
 3550 LX=LC+L(IND1)
      IND2=LX-LIM1
      IF(IND2) 3570,3560,3560
 3560 LX=IND2
 3570 LX=LX+1
      IR1(I)=EOR(IA(LX),IR1(I))
 3620 IND3=I+1
      IR3(I)=IR1(IND3)
 3700 CONTINUE
      LN=N-LN+1
      IDSTA=IDN
      GO TO 2200
C   PREPARE FOR CHIEN SEARCH
 5000 LIM4=0
      DO 5030 I=1,NERRO
      IND1=LIM6-I
      IF(IR1(IND1))5020,5030,5020
 5020 LIM4=LIM4+1
      ISIG(LIM4)=I
      IND2=IR1(IND1)
      LSIG(LIM4)=L(IND2)
 5030 CONTINUE
      IF(LIM4) 8000,8000,5040
 5040 NERR=0
C   CHIEN SEARCH
      DO 5900 I=1,LIM5
      IDEST=1
      DO 5200 J=1,LIM4
      LX=ISIG(J)+LSIG(J)
      IND1=LX-LIM1
      IF(IND1) 5110,5100,5100
 5100 LX=IND1
```

```
5110 LSIG(J)=LX
     IND1=LX+1
     ITEST=EOR(ITEST,IA(IND1))
5200 CONTINUE
     IF(ITEST)5900,5210,5900
5210 NERR=NERR+1
     LOC(NERR)=I
5900 CONTINUE
     IF(NERR) 8000,8000,6000
C  COMPLTE COEF. OF A POLY.
6000 LIM6=0
     DO 6050 I=1,NERRO
     IND1=I-1
     IAX=IS(I)
     IF(IND1) 6020,6041,6020
6020 DO 6040 K=1,IND1
     IND2=LIM8+K-I
     IND3=IR1(IND2)
     IND4=IS(K)
     IF(IND4) 6030,6040,6030
6030 IF(IND3) 6031,6040,6031
6031 LX=L(IND3)+L(IND4)
     IND2=LX-LIM1
     IF(IND2) 6034,6032,6032
6032 LX=IND2
6034 LX=LX+1
     IAX=EOR(IAX,IA(LX))
6040 CONTINUE
6041 IF(IAX) 6042,6050,6042
6042 LIM6=LIM6+1
     LAA(LIM6)=L(IAX)
     IAA(LIM6)=IND1
6050 CONTINUE
C  COMPLTE COEF. OF SIGMA PRIME POLY.
     LIM7=0
     DO 6100 I=1,NERRO,2
     IND1=NERRO-I+1
     IF(IR1(IND1,)) 6060,6100,6060
6060 LIM7=LIM7+1
     ISIGP(LIM7)=I-1
```

```
      IND2=IR1(IND1)
      LSIGP(LIM7)=L(IND2)
 6100 CONTINUE
C  COMPUTE ERROR VALUE
      DO 7000 I=1,NERR
      JA=0
      DO 6200 J=1,LIM6
      LX=LAA(J)+LOC(I)*JAA(J)
      LX=LX-(LX/LIM1)*LIM1+1
      JA=EOR(JA,IA(LX))
 6200 CONTINUE
      JSIGP=0
      DO 6300 J=1,LIM7
      LX=LSIGP(J)+LOC(I)*ISIGP(J)
      LX=LX-(LX/LIM1)*LIM1+1
      JSIGP=EOR(JSIGP,IA(LX))
 6300 CONTINUE
      LX=L(JA)-L(JSIGP)
      IF(LX)6400,6410,6410
 6400 LX=LX+LIM1
 6410 IND1=LX+1
      IVALU(I)=IA(IND1)
 7000 CONTINUE
 8000 WRITE(5,8100) KO,NERRC
 8100 FORMAT(12H1A 2 TO THE ,I2,17H SYMBOL R-S CODE.
     1/32H NUMBER OF CORRECTABLE ERRORS = ,I2/////)
      IF(LIM4) 8115,8115,8110
 8110 IF(NERR) 8115,8115,8130
 8115 WRITE(5,8120)
 8120 FORMAT(10H NO ERRORS)
      GO TO 9000
 8130 WRITE(5,8140)
 8140 FORMAT(34H      ERROR LOCATION    ERROR VALUE/)
      DO 8300 I=1,NERR
      WRITE(5,8200) I,LOC(I),IVALU(I)
 8200 FORMAT(1H ,I2,7X,I5,10X,I5)
 8300 CONTINUE
 9000 STOP
      END
```

The following pages give the assembly language version of the preceding program. It is organized so that the input and output statements are in Fortran and the rest of the program is in an assembly language subprogram.

```
      DIMENSION INPL(1),JNPL(15),INPV(1),JNPV(15),LOC(1),JLOC(15)
     1,IVALU(1),JVALU(15),IF(1),JF(6)
      DATA JNPL/211,203,200,122,121,120,111,101,42,41,30,20,14,7,5/
      DATA JNPV/13,13,92,91,250,201,111,6,51,77,101,101,7,6,2/
      DATA JF/0,0,0,1,1,1/
      IF(1)=0
      INPL(1)=2
      INPV(1)=1
      JLOC(1)=0
      JVALU(1)=0
      K0=8
      NERRO=16
      NEACT=16
      CALL RS(INPL,INPV,IF,K0,NERRO,NEACT,LIM4,NERR,LOC,IVALU)
      WRITE(5,8100) K0,NERRO
 8100 FORMAT(12H A 2 TO THE ,I2,17H SYMBOL R-S CODE.
     1/32H NUMBER OF CORRECTABLE ERRORS = ,I2/////)
      IF(LIM4) 8115,8115,8110
 8110 IF(NERR) 8115,8115,8130
 8115 WRITE(5,8120)
 8120 FORMAT(10H NO ERRORS)
      GO TO 9000
 8130 WRITE(5,8140)
 8140 FORMAT(34H       ERROR LOCATION   ERROR VALUE/)
      DO 8300 I=1,NERR
      J=NERR+1-I
      WRITE(5,8200) I,JLOC(J),JVALU(J)
 8200 FORMAT(1H ,I2,7X,I5,10X,I5)
 8300 CONTINUE
 9000 STOP
      END
```

```
0000      19880000 00001            ENT      RS
0000 0    0000     00002      RS    DC       0
0001 01   C4800000 00003            LD    I  RS
0003 01   D40003AD 00004            STO   L  A1+1
0005 01   74010000 00005            MDM   L  RS,1
0007 01   C4800000 00006            LD    I  RS
0009 01   D40003A1 00007            STO   L  B1+1
000B 01   74010000 00008            MDM   L  RS,1
000D 01   C4800000 00009            LD    I  RS
000F 01   D4000366 00010            STO   L  C1+1
0011 01   74010000 00011            MDM   L  RS,1
0013 01   C4800000 00012            LD    I  RS
0015 01   D400043D 00013            STO   L  K0
0017 01   C480043D 00014            LD    I  K0
0019 01   D400043F 00015            STO   L  K0
001B 01   74010000 00016            MDM   L  RS,1
001D 01   C4800000 00017            LD    I  RS
001F 01   D400043E 00018            STO   L  NERRO
0021 01   C480043E 00019            LD    I  NERRO
0023 01   D400043E 00020            STO   L  NERRO
0025 01   74010000 00021            MDM   L  RS,1
0027 01   C4800000 00022            LD    I  RS
0029 01   D400043F 00023            STO   L  NEACT
002B 01   C480043F 00024            LD    I  NEACT
002D 01   D400043F 00025            STO   L  NEACT
002F 01   74010000 00026            MDM   L  RS,1
0031 01   C4800000 00027            LD    I  RS
0033 01   D4000440 00028            STO   L  LIM4
0035 01   74010000 00029            MDM   L  RS,1
0037 01   C4800000 00030            LD    I  RS
0039 01   D4000441 00031            STO   L  NERR
003B 01   74010000 00032            MDM   L  RS,1
003D 01   C4800000 00033            LD    I  RS
003F 01   D40004ED 00034            STO   L  D1+1
0041 01   D4000580 00035            STO   L  D2+1
0043 01   D400059D 00036            STO   L  D3+1
0045 01   74010000 00037            MDM   L  RS,1
0047 01   C4800000 00038            LD    I  RS
0049 01   D40005B8 00039            STO   L  E1+1
004B 01   74010000 00040            MDM   L  RS,1
004D 01   4C0002FC 00041            B     L  L00
```

```
004F      0008      00042   IG     BSS      8
0057      00FF      00043   L      BSS      255
0156      00FF      00044   IA     BSS      255
0255      0021      00045   IS     BSS      33
0276      0011      00046   IR1    BSS      17
0287      0010      00047   IR2    BSS      16
0297      0010      00048   IR3    BSS      16
02A7      0010      00049   ISIG   BSS      16
02B7      0010      00050   LSIG   BSS      16
02C7      0010      00051   IAA    BSS      16
02D7      0010      00052   LAA    BSS      16
02E7      0009      00053   ISIGF  BSS      9
02F0      0009      00054   LSIGP  BSS      9
02FA      0002      00055   DCLB   BSS   E  2
                    00056   *      CRA      2
02FC 0  0544        00057   L00    DC       /0544
02FD 01 D400043B    00058          STO   L  REG2
                    00059   *      CPA      3
02FF 0  0546        00060          DC       /0546
0300 01 D400043C    00061          STO   L  REG3
0302 01 C400042A    00062          LD    L  ONE
0304 01 6580043D    00063          LDX   I1 K0
0306 0  1100        00064          SLA    1
0307 01 9400042A    00065          S     L  ONE
0309 01 D400042C    00066          STO   L  LIM1
030B 01 C400043E    00067          LD    L  NERRO
030D 01 8400042A    00068          A     L  ONE
030F 01 D400042D    00069          STO   L  LIM8
0311 01 94000429    00070          S     L  TWO
0313 01 D400042E    00071          STO   L  LIM2
0315 01 C400043E    00072          LD    L  NERRO
0317 0  1001        00073          SLA      1
0318 01 D4000430    00074          STO   L  LIM9
031A 01 8400042A    00075          A     L  ONE
031C 01 D40004FD    00076          STO   L  IND1
                    00077   *      CAR      1
031E 0  0562        00078          DC       /0562
031F 01 94000429    00079          S     L  TWO
0321 01 D400042F    00080          STO   L  LM1
0323 01 C400042D    00081          LD    L  ZERO
```

```
0325 01 D5000254 00082          STO  L1 IS-1
0327 01 C400043D 00083          LD   L  KO
0329 01 9400042A 00084          S    L  ONE
032B 01 D4000431 00085          STO  L  LIM3
032D 01 C400042C 00086          LD   L  LIM1
032F 01 94000430 00087          S    L  LIM9
0331 01 D40004FF 00088          STO  L  LIM5
0333 01 6580042D 00089          LDX  I1 LIM8
0335 01 C400042A 00090          LD   L  ONE
0337 01 D5000275 00091          STO  L1 IR1-1
0339 01 D400004F 00092          STO  L  IG
033B 01 C400043D 00093          LD   L  KO
                 00094    *     CAR     4
033D 0  056E     00095          DC      /056E
033E 01 6580042A 00096          LDX  I1 ONE
0340 01 C400042B 00097          LD   L  ZERO
0342 01 D500004F 00098    L60   STO  L1 IG
                 00099    *     ICL1    L60
0344 0  39FC     00100          DC      /3A00+L60-*
0345 01 D4000057 00101          STO  L  L
0347 01 C400042A 00102          LD   L  ONE
0349 01 D400015E 00103          STO  L  IA
034B 01 D4000432 00104          STO  L  LXP1
034D 01 6580043D 00105    L100  LDX  I1 KO
034F 01 C500004E 00106          LD   L1 IG-1
0351 01 D4000433 00107          STO  L  IGL
0353 01 C400042B 00108          LD   L  ZERO
0355 01 D4000434 00109          STO  L  IAX
0357 01 C400043D 00110          LD   L  KO
                 00111    *     CAR     4
0359 0  056E     00112          DC      /056E
035A 01 6580042A 00113          LDX  I1 ONE
035C 01 C400043D 00114    L140  LD   L  KO
                 00115    *     SBR     1
035E 0  0082     00116          DC      /0082
                 00117    *     CAR     2
035F 0  0564     00118          DC      /0564
0360 01 9400042A 00119          S    L  ONE
                 00120    *     CAR     3
0362 0  0566     00121          DC      /0566
```

```
0363 01 C4000433 00122          LD   L  IGL
0365 01 A7000367 00123    C1    M    L3 *
0367 01 C700004F 00124          LD   L3 IG
                 00125    *     EORR    0
0369 0  388A     00126          DC      /388A
036A 01 D600004F 00127          STO  L2 IG
036C 01 4C080375 00128          BNP     L200
036E 01 C400042A 00129          LD   L  ONE
0370 0  1200     00130          SLA  2
0371 01 84000434 00131          A    L  IAX
0373 01 D4000434 00132          STO  L  IAX
                 00133    *     ICL1    L140
0375 0  39F6     00134    L200  DC      /3A00+L140-*
0376 01 C4000434 00135          LD   L  IAX
0378 01 84000433 00136          A    L  IGL
037A 01 D4000434 00137          STO  L  IAX
037C 01 C4000433 00138          LD   L  IGL
037E 01 D400004F 00139          STO  L  IG
0380 01 C4000432 00140          LD   L  LXP1
0382 01 65800434 00141          LDX  I1 IAX
0384 01 D5000056 00142          STO  L1 L-1
                 00143    *     CAR     2
0386 0  0564     00144          DC      /0564
0387 01 8400042A 00145          A    L  ONE
0389 01 D4000432 00146          STO  L  LXP1
038B 01 C4000434 00147          LD   L  IAX
038D 01 D6000156 00148          STO  L2 IA
038F 01 C4000432 00149          LD   L  LXP1
0391 01 9400042C 00150          S    L  LIM1
0393 01 4C28034D 00151          BN      L100
0395 01 C400043F 00152          LD   L  NEACT
                 00153    *     CAR     4
0397 0  056E     00154          DC      /056E
0398 01 6580042A 00155          LDX  I1 ONE
039A 01 C400042B 00156    L300  LD   L  ZERO
039C 01 D5000254 00157          STO  L1 IS-1
039E 01 6680042B 00158          LDX  I2 ZERO
03A0 01 C60003A2 00159    B1    LD   L2 *
                 00160    *     CAR     3
03A2 0  0566     00161          DC      /0566
```

PAGE   5

```
03A3 01 C7000056 00162          LD    L3 L-1
                 00163    *     CAR      Q
03A5 0  056A     00164          DC       /056A
03A6 01 C400042B 00165          LD    L  ZERO
03A8 01 DC0002FA 00166          STD   L  DOUB
03AA 01 C400042C 00167          LD    L  LIM1
03AC 01 960003AE 00168    A1    S     L2 *
03AE 01 D40004FE 00169          STO   L  IND2
                 00170    *     CRA      1
03B0 0  0542     00171          DC       /0542
03B1 01 A40004FE 00172          M     L  IND2
03B3 01 8C0002FA 00173          AD    L  DOUB
03B5 01 AC00042C 00174          D     L  LIM1
03B7 0  109C     00175          SLT      16
                 00176    *     CAR      3
03B8 0  0566     00177          DC       /0566
03B9 01 C7000156 00178          LD    L3 IA
                 00179    *     CAR      8
03BB 0  0576     00180          DC       /0576
03BC 01 C5000254 00181          LD    L1 IS-1
                 00182    *     EORR     8
03BE 0  3896     00183          DC       /3896
03BF 01 D5000254 00184          STO   L1 IS-1
                 00185    *     ICL2     B1
03C1 0  3ACE     00186          DC       /3B00+B1-*
                 00187    *     IN       1
03C2 0  05C2     00188          DC       /05C2
                 00189    *     CRA      1
03C3 0  0542     00190          DC       /0542
03C4 01 94000430 00191          S     L  LIM9
03C6 01 4C08039A 00192          BNP      L300
03C8 01 C4000442 00193          LD    L  NTIME
                 00194    *     CAR      G
03CA 0  057E     00195          DC       /057E
03CB 0  3000     00196          WAIT
03CC 01 658C042F 00197    L305  LDX   I1 LIM2
03CE 0  C05E     00198          LD       ONE
03CF 0  D065     00199          STO      IDSTA
03D0 01 D5000297 00200          STO   L1 IR3
03D2 0  C058     00201          LD       ZERO
```

```
03C3 0  D062      00202          STO     N
03C4 0  D062      00203          STO     LN
03C5 01 D5000206  00204          STO  L1 IR1
03C7 01 D5000287  00205          STO  L1 IR2
                  00206    *     CAR     2
03C9 0  0564      00207          DC      /0564
                  00208    *     X1R     4
03CA 0  018E      00209          DC      /018E
03CB 01 D6000276  00210  L310    STO  L2 IR1
03CD 01 D6000287  00211          STO  L2 IR2
03CF 01 D6000297  00212          STO  L2 IR3
                  00213    *     TCL2    L310
03E1 0  3AF5      00214          DC      /3800+L310-*
03E2 01 65800436  00215  L1000   LDX  I1 N
03E4 01 C5000255  00216          LD   L1 IS
                  00217    *     CAR     9
03E6 0  0578      00218          DC      /0578
03E7 01 6580042B  00219          LDX  I1 ZERO
03E9 0  C054      00220          LD      NERRO
                  00221    *     CAR     4
03EA 0  056E      00222          DC      /056E
03EB 01 C5000276  00223  L1100   LD   L1 IR1
03ED 01 4C180402  00224          BZ      L1900
                  00225    *     CAR     2
03EF 0  0564      00226          DC      /0564
03F0 01 C5000287  00227          LD   L1 IR2
03F2 01 4C180402  00228          BZ      L1900
                  00229    *     CAR     3
03F4 0  0566      00230          DC      /0566
03F5 01 C6000056  00231          LD   L2 L-1
03F7 01 87000056  00232          A    L3 L-1
                  00233    *     CAR     2
03F9 0  0564      00234          DC      /0564
03FA 0  9031      00235          S       LIM1
03FB 01 4C2803FE  00236          BN      L1300
                  00237    *     CAR     2
03FD 0  0564      00238          DC      /0564
03FE 01 C6000156  00239  L1300   LD   L2 IA
                  00240    *     EORK    9
0400 0  3898      00241          DC      /3898
```

```
                        00242    *     CAR       9
0401 0  0578            00243          DC        /0578
                        00244    *     ICL1      L1100
0402 0  39F8            00245    L1900 DC        /3A00+L1100-*
                        00246    *     CRA       9
0403 0  0558            00247          DC        /0558
0404 01 4C200443        00248          BNZ       L3000
0406 0  C027            00249    L2010 LD        LIM2
                        00250    *     CAR       4
0407 0  056E            00251          DC        /056E
0408 01 6580042B        00252          LDX   I1  ZERO
040A 01 C5000298        00253    L2050 LD    L1  IR3+1
040C 01 D5000297        00254          STO   L1  IR3
                        00255    *     ICL1      L2050
040E 0  39FE            00256          DC        /3A00+L2050-*
040F 0  C01E            00257          LD        ZERO
0410 01 D5000297        00258          STO   L1  IR3
0412 0  C023            00259    L2200 LD        N
0413 0  901E            00260          S         LM1
0414 01 4C1804A2        00261          BZ        L5000
0416 0  C017            00262    L2300 LD        LIM2
                        00263    *     CAR       4
0417 0  056E            00264          DC        /056E
0418 01 6580042B        00265          LDX   I1  ZERO
041A 01 C5000288        00266    L2350 LD    L1  IR2+1
041C 01 D5000287        00267          STO   L1  IR2
                        00268    *     ICL1      L2350
041E 0  39FB            00269          DC        /3A00+L2350-*
041F 01 66800436        00270          LDX   I2  N
0421 01 C6000255        00271          LD    L2  IS
0423 01 D5000287        00272          STO   L1  IR2
0425 0  C010            00273          LD        N
0426 0  8003            00274          A         ONE
0427 0  D00E            00275          STO       N
0428 0  70B9            00276          B         L1000
0429 0  0002            00277    TWO   DC        2
042A 0  0001            00278    ONE   DC        1
042B 0  0000            00279    ZERO  DC        0
042C 0  0000            00280    LIM1  DC        0
042D 0  0000            00281    LIM8  DC        0
```

```
042E 0  0000      00282   LIM2  DC      0
042F 0  0000      00283   LM1   DC      0
0430 0  0000      00284   LIM9  DC      0
0431 0  0000      00285   LIM3  DC      0
0432 0  0000      00286   LXF1  DC      0
0433 0  0000      00287   IGL   DC      0
0434 0  0000      00288   IAX   DC      0
0435 0  0000      00289   IDSTA DC      0
0436 0  0000      00290   N     DC      0
0437 0  0000      00291   LN    DC      0
0438 0  0000      00292   LD    DC      0
0439 0  0000      00293   ICN   DC      0
043A 0  0000      00294   LIM7  DC      0
043B 0  0000      00295   REG2  DC      0
043C 0  0000      00296   REG3  DC      0
043D 0  0000      00297   KO    DC      0
043E 0  0000      00298   NERRO DC      0
043F 0  0000      00299   NEACT DC      0
0440 0  0000      00300   LIM4  DC      0
0441 0  0000      00301   NERR  DC      0
0442 0  03E7      00302   NTIME DC      999
0443 0  C0F3      00303   L3000 LD      LN
0444 0  1001      00304         SLA     1
0445 0  90F0      00305         S       N
0446 01 4C80046F  00306         BNP     L3500
                  00307   *     CRA     9
0448 0  0558      00308   L3100 DC      /0558
                  00309   *     CAR     2
0449 0  0564      00310         DC      /0564
044A 01 67000435  00311         LDX   I3 IDSTA
044C 01 C6000056  00312         LD    L2 L-1
044E 01 97000056  00313         S     L3 L-1
0450 01 4C100453  00314         BNN     L3310
0452 0  80D9      00315         A       LIM1
0453 0  D0F4      00316   L3310 STO     LD
0454 0  C0E9      00317         LD      NERRO
                  00318   *     CAR     4
0455 0  056F      00319         DC      /056F
0456 01 6580042B  00320         LDX   I1 ZERO
0456 01 C5000297  00321   L3320 LD    L1 IK3
```

```
045A 01 4C18046D 00322            BZ        L3450
                 00323      *     CAR       2
045C 0  0564     00324            DC        /0564
045D 01 C6000056 00325            LD     L2 L-1
045F 0  80C8     00326            A         LD
                 00327      *     CAR       2
0460 0  0564     00328            DC        /0564
0461 0  90CA     00329            S         LIM1
0462 01 4C280465 00330            BN        L3340
                 00331      *     CAR       2
0464 0  0564     00332            DC        /0564
0465 01 C6000156 00333   L3340    LD     L2 IA
                 00334      *     CAR       7
0467 0  0574     00335            DC        /0574
0468 01 C5000276 00336            LD     L1 IR1
                 00337      *     EORR      7
046A 0  3894     00338            DC        /3894
046B 01 D5000276 00339            STO    L1 IR1
                 00340      *     ICL1      L3320
046D 0  39FA     00341   L3450    DC        /3A00+L3320-*
046E 0  7097     00342            B         L2010
                 00343      *     CRA       9
046F 0  0558     00344   L3500    DC        /0558
0470 0  D0C8     00345            STO       IDN
                 00346      *     CAR       2
0471 0  0564     00347            DC        /0564
0472 01 67800435 00348            LDX    I3 IDSTA
0474 01 C6000056 00349            LD     L2 L-1
0476 01 97000056 00350            S      L3 L-1
0478 01 4C10047B 00351            BNN       L3540
047A 0  80E1     00352            A         LIM1
047B 0  D0FC     00353   L3540    STO       LD
047C 0  C0C1         4            LD        NERRO
                     5      *     CAR       4
047D 0  056E     00356            DC        /056E
047E 01 6580042B 00357            LDX    I1 ZERO
0480 01 C5000297 00358   L3550    LD     L1 IR3
0482 01 4C180495 00359            BZ        L3620
                 00360      *     CAR       2
```

```
0484 0  0564      00361           DC        /0564
0485 01 C60000F3  00362           LD    L2  L-1
0487 0  80E0      00363           A         LD
                  00364     *     CAP       2
0488 0  0564      00365           DC        /0564
0489 0  90A2      00366           S         LIM1
048A 01 4C28048D  00367           BN        L3570
                  00368     *     CAR       2
048C 0  0564      00369           DC        /0564
048D 01 C6000156  00370   L3570   LD    L2  IA
                  00371     *     CAR       7
048F 0  0574      00372           DC        /0574
0490 01 C5000276  00373           LD    L1  IR1
                  00374     *     EORR      7
0492 0  3894      00375           DC        /3894
0493 01 D5000276  00376           STO   L1  IR1
0495 01 C5000277  00377   L3620   LD    L1  IR1+1
0497 01 D5000297  00378           STO   L1  IR3
                  00379     *     ICL       L3550
0499 0  39E6      00380           DC        /3A00+L3550-*
049A 0  C09F      00381           LD        IDK
049B 0  D099      00382           STO       IDSTA
049C 0  C099      00383           LD        N
049D 0  800C      00384           A         ONE
049E 0  9098      00385           S         LN
049F 0  D097      00386           STO       LN
04A0 01 4C000412  00387           B     L   L2200
                  00388     *     DERS      G
04A2 0  06FE      00389   L5000   DC        /06FE
04A3 0  7002      00390           B         LLLA
04A4 01 4C0003CC  00391           B     L   L305
04A6 0  3000      00392   LLLA    WAIT
04A7 0  C09A      00393           LD        NTIME
                  00394     *     CAR       G
04A8 0  057E      00395           DC        /057E
04A9 01 6680042B  00396   L3601   LDX   I2  ZERO
04AB 01 C400043E  00397           LD    L   NERFO
                  00398     *     CAR       4
04AD 0  056E      00399           DC        /056E
04AE 01 6580042B  00400           LDX   I1  ZERO
```

```
04BC 01 C400043E 00401        LD   L  NERRO
04B2 0  D04A     00402        STO     IND1
04B3 0  C049     00403  L5020 LD      IND1
04B4 0  90CE     00404        S       ONE1
04B5 0  D047     00405        STO     IND1
                 00406  *     CAR     3
04B6 0  0566     00407        DC      /0566
04B7 01 C7000276 00408        LD   L3 IR1
04B9 01 4C1804C5 00409        BZ      L5030
                 00410  *     CAR     3
04BB 0  0566     00411        DC      /0566
04BC 01 C7000056 00412        LD   L3 L-1
04BE 01 D60002B7 00413        STO  L2 LSIG
                 00414  *     CRA     1
04C0 0  0542     00415        DC      /0542
04C1 0  8035     00416        A       ONE1
04C2 01 D60002A7 00417        STO  L2 ISIG
                 00418  *     IN      2
04C4 0  05C4     00419        DC      /05C4
                 00420  *     ICL1    L5020
04C5 0  39FD     00421  L5030 DC      /3A00+L5020-*
                 00422  *     CRA     2
04C6 0  0544     00423        DC      /0544
04C7 01 D4800440 00424        STO  I  LIM4
04C9 01 4C0805C3 00425        BNP     L8000
04CB 0  C02F     00426        LD      ONE1
04CC 0  D034     00427        STO     I
04CD 01 6780042B 00428        LDX  I3 ZERO
04CF 01 C4800440 00429        LD   I  LIM4
                 00430  *     CAR     4
04D1 0  056E     00431        DC      /056E
04D2 0  C028     00432  L5050 LD      ONE1
                 00433  *     CAR     7
04D3 0  0574     00434        DC      /0574
04D4 01 6580042B 00435        LDX  I1 ZERO
04D6 01 C50002A7 00436  L5060 LD   L1 ISIG
04D8 01 850002B7 00437        A    L1 LSIG
                 00438  *     CAR     2
```

-127

```
04DA 0  0564      00439            DC       /0564
04DB 01 9400042C  00440            S     L  LIN1
04DD 01 4C2804E0  00441            BN       L5110
                  00442      *     CAR      2
04DF 0  0564      00443            DC       /0564
                  00444      *     CRA      2
04E0 0  0544      00445      L5110 DC       /0544
04E1 01 D5000267  00446            STO   L1 LSIG
04E3 01 C6000156  00447            LD    L2 IA
                  00448      *     EORR     7
04E5 0  3894      00449            DC       /3894
                  00450      *     CAR      7
04E6 0  0574      00451            DC       /0574
                  00452      *     ICL1     L5060
04E7 0  39EE      00453            DC       /3A00+L5060-*
                  00454      *     CRA      7
04E8 0  0554      00455            DC       /0554
04E9 01 4C2004EF  00456            BNZ      L5900
04EB 0  C015      00457            LD       I
04EC 01 D70004EE  00458      D1    STO   L3 *
                  00459      *     IN       3
04EE 0  05C6      00460            DC       /05C6
04EF 0  C011      00461      L5900 LD       I
04F0 0  800A      00462            A        ONE1
04F1 0  D00F      00463            STO      I
04F2 0  900C      00464            S        LIME
04F3 01 4C2804D2  00465            BNP      L5050
                  00466      *     CRA      3
04F5 0  0546      00467            DC       /0546
04F6 01 D4000441  00468            STO   I  NERR
04F8 01 4C0805C3  00469            BNP      L8000
04FA 0  7007      00470            B        L5950
04FB 0  0001      00471      ONE1  DC       1
04FC 0  0000      00472      ZERO1 DC       0
04FD 0  0000      00473      IND1  DC       0
04FE 0  0000      00474      IND2  DC       0
04FF 0  0000      00475      LIM5  DC       0
0500 0  0000      00476      LIM6  DC       0
0501 0  0000      00477      *     DC       0
                  00478      *     DER      G
```

```
0502 0  07FE      00479    L5950 DC        /07FE
0503 0  70AE      00480          B         L5001
0504 0  3000      00481          WAIT
0505 01 C4000442  00482          LD    L   NTIME
                  00483    *     CAR       G
0507 0  057E      00484          DC        /057E
0508 0  C0F3      00485    L5999 LD        ZERO1
0509 0  D0F6      00486          STO       LIM6
050A 0  D0F6      00487          STO       I
050B 01 66800501  00488    L6000 LDX   I2  I
050D 01 C6000255  00489          LD    L2  IS
                  00490    *     CAR       7
050F 0  0574      00491          DC        /0574
0510 0  C0F0      00492          LD        I
0511 01 4C180538  00493          BZ        L6041
                  00494    *     CAR       4
0513 0  056E      00495          DC        /056E
0514 01 C400043E  00496          LD    L   NERRO
0516 0  90EA      00497          S         I
0517 0  D0F6      00498          STO       IND2
0518 01 6580042B  00499          LDX   I1  ZERO
051A 01 668004FE  00500    L6020 LDX   I2  IND2
051C 01 C6000276  00501          LD    L2  IR1
051E 01 4C180534  00502          BZ        L6040
                  00503    *     CAR       2
0520 0  0564      00504          DC        /0564
0521 01 C5000255  00505          LD    L1  IS
0523 01 4C180534  00506          BZ        L6040
                  00507    *     CAR       3
0525 0  0566      00508          DC        /0566
0526 01 C6000056  00509          LD    L2  L-1
0528 01 870C0056  00510          A     L3  L-1
                  00511    *     CAR       2
052A 0  0564      00512          DC        /0564
052B 01 9400042C  00513          S     L   LIM1
052D 01 4C280530  00514          BN        L6034
                  00515    *     CAR       2
052F 0  0564      00516          DC        /0564
0530 01 C6000156  00517    L6034 LD    L2  IA
                  00518    *     EORR      7
```

```
0532 0  3894      00519          DC        /3894
                  00520    *     CAR       7
0533 0  0574      00521          DC        /0574
0534 0  C0C9      00522  L6040   LD        IND2
0535 0  80C5      00523          A         ONE1
0536 0  D0C7      00524          STO       IND2
                  00525    *     ICL1      L6020
0537 0  39E2      00526          DC        /3A00+L6020-*
                  00527    *     CRA       7
0538 0  0554      00528  L6041   DC        /0554
0539 01 4C180547  00529          BZ        L6050
                  00530    *     CAR       2
053B 0  0564      00531          DC        /0564
053C 0  C0C3      00532          LD        LIN6
                  00533    *     CAR       3
053D 0  0566      00534          DC        /0566
053E 0  80BC      00535          A         ONE1
053F 0  D0C0      00536          STO       LIN6
0540 01 C6000056  00537          LD    L2  L-1
0542 01 D70002D7  00538          STO   L3  LAA
0544 0  C0BC      00539          LD        I
0545 01 D70002C7  00540          STO   L3  IAA
0547 0  C0B9      00541  L6050   LD        I
0548 0  80B2      00542          A         ONE1
0549 0  D0B7      00543          STO       I
054A 01 9400042E  00544          S     L   LIN2
054C 01 4C08050B  00545          BNP       L6000
054E 01 6680042B  00546          LDX   I2  ZERO
0550 0  C0AE      00547          LD        ZERO1
0551 0  D0AF      00548          STO       I
0552 01 C400043E  00549  L6060   LD    L   NERRO
0554 0  90AC      00550          S         I
                  00551    *     CAR       1
0555 0  0562      00552          DC        /0562
0556 01 C5000275  00553          LD    L1  IR1-1
0558 01 4C180563  00554          BZ        L6100
                  00555    *     CAR       1
055A 0  0562      00556          DC        /0562
055B 01 C5000056  00557          LD    L1  L-1
055D 01 D60002FC  00558          STO   L2  LSIGP
```

```
055F 0  C0A1      00559              LD      I
0560 01 D60002E7  00560              STO  L2 ISIGP
                  00561       *      IN      2
0562 0  05C4      00562              DC      /05C4
0563 0  C09D      00563  L6100 LD      I
0564 01 84000429  00564              A    L  TWO
0566 0  D09A      00565              STO     I
0567 01 9400043E  00566              S    L  NEPRO
0569 01 4C080552  00567              BNP     L6060
                  00568       *      CRA     2
056B 0  0544      00569              DC      /0544
056C 01 D400043A  00570              STO  L  LIM7
056E 01 6580042B  00571              LDX  I1 ZERO
0570 01 C400042B  00572  L6150 LD    L  ZERO
                  00573       *      CAR     8
0572 0  0576      00574              DC      /0576
0573 01 C4000500  00575              LD   L  LIM6
                  00576       *      CAR     4
0575 0  056E      00577              DC      /056E
0576 01 6680042B  00578              LDX  I2 ZERO
0578 01 C60002D7  00579  L6200 LD    L2 LAA
                  00580       *      CAR     Q
057A 0  056A      00581              DC      /056A
057B 01 C400042B  00582              LD   L  ZERO
057D 01 DC0002FA  00583              STD  L  DOUB
057F 01 C5000581  00584  D2    LD    L1 *
0581 01 A60002C7  00585              M    L2 IAA
0583 01 8C0002FA  00586              AD   L  DOUB
0585 01 AC00042C  00587              D    L  LIM1
                  00588       *      X3R     Q
0587 0  038A      00589              DC      /038A
0588 01 C7000156  00590              LD   L3 IA
                  00591       *      EORK    8
058A 0  3896      00592              DC      /3896
                  00593       *      CAR     8
058B 0  0576      00594              DC      /0576
                  00595       *      ICL2    L6200
058C 0  3ADE      00596              DC      /3D00+L6200-*
058D 01 C400042B  00597              LD   L  ZERO
                  00598       *      CAR     E
```

```
058F 0  057A      00599           DC        /057A
0590 01 C400043A  00600           LD     L  LIM7
                  00601     *     CAR       4
0592 0  056E      00602           DC        /056E
0593 01 6680042B  00603           LDX    I2 ZERO
0595 01 C60002F0  00604     L630C LD     L2 LSTGP
                  00605     *     CAR       Q
0597 0  056A      00606           DC        /056A
0598 01 C400042B  00607           LD     L  ZERO
059A 01 DC0002FA  00608           STD    L  DOUP
059C 01 C500059E  00609     D3    LD     L1 *
059E 01 A60002F7  00610           M      L2 ISTGP
05A0 01 8C0002FA  00611           AD     L  DOUP
05A2 01 AC00042C  00612           D      L  LIM1
                  00613     *     X3R       Q
05A4 0  038A      00614           DC        /038A
05A5 01 C7000156  00615           LD     L3 IA
                  00616     *     ECRR      E
05A7 0  389A      00617           DC        /389A
                  00618     *     CAR       E
05A8 0  057A      00619           DC        /057A
                  00620     *     ICL2      L6300
05A9 0  3AEP      00621           DC        /3B00+L6300-*
                  00622     *     X2R       8
05AA 0  0296      00623           DC        /0296
                  00624     *     X3R       E
05AB 0  039A      00625           DC        /039A
05AC 01 C6000056  00626           LD     L2 L-1
05AE 01 97000056  00627           S      L3 L-1
05B0 01 4C1005B4  00628           BNN       L6410
05B2 01 8400042C  00629           A      L  LIM1
                  00630     *     CAR       2
05B4 0  0564      00631     L6410 DC        /0564
05B5 01 C6000156  00632           LD     L2 IA
05B7 01 D50005B9  00633     E1    STO    L1 *
                  00634     *     IN        1
05B9 0  05C2      00635           DC        /05C2
                  00636     *     CRA       1
05BA 0  0542      00637           DC        /0542
05BD 01 94800441  00638           S      I  NERR
```

```
05BD 01 4C200570 00639           BNZ      L6150
05BF 01 6680043B 00640           LDX   I2 REG2
05C1 01 6780043C 00641           LDX   I3 REG3
                 00642     *     DERS     G
05C3 0  06FE     00643     L8000 DC       /06FE
05C4 0  7002     00644           B        LLLB
05C5 01 4C00050B 00645           B     L  L5999
05C7 0  3000     00646     LLLB  WAIT
05C8 01 4C800000 00647           B     I  RS
05CA             00648           END
```

REFERENCES

1. P. Elias, "Error-free Coding," IEEE Trans. on Information Theory, Vol IT-4, 1954.

2. G. D. Forney, Concatenated Codes, MIT Press, Cambridge, Mass. 1966.

3. M. S. Pinsker, "On the Complexity of Decoding," Prob. Peredachi Information, Vol. I, No. 1, pp 84-86, 1965.

4. I. G. Stiglitz, "Iterative Sequential Decoding," IEEE Trans. on Information Theory, Vol. IT-15, pp. 715-721, Nov. 1969.

5. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," Journ. S.I.A.M., Vol. 8, 1960.

6. J. P. Odenwalder, "Optimal Decoding of Convolutional Codes," PhD Dissertation, School of Engineering and Applied Science, University of California, Los Angeles, March, 1972.

7. J. A. Heller, "Short Constraint Length Convolutional Codes," Jet Propulsion Laboratory, California Institute of Technology, Space Programs Summary 37-54, Vol. III, pp. 171-177, Oct./Nov.1968.

8. LINKABIT Corporation, "Coding Systems Study for High Data Rate Telemetry Links," Final Report on Contract NAS2-6024, NASA Ames Research Center, Moffett Field, California, Jan.1971.

9. LINKABIT Corporation, "Performance Study of Viterbi Decoding as Related to Space Communications," Final Report on Contract DAAE07-71-0148, USASATCOMA, Fort Monmouth, New Jersey, (Unclassified), Jan.1972.

10. R. D. Gallager, Information Theory and Reliable Communication, New York: Wiley, 1968.

11. E. R. Berlekamp, Algebraic Coding Theory, New York: McGraw-Hill, 1968.

12. J. L. Massey, "Shift-Register Synthesis and BCH Decoding," IEEE Trans. on Information Theory, Vol. IT-15, pp 122-127, Jan. 1969.

13. T. C. Bartee and D. I. Schneider, "Computation with Finite Fields"Information and Control, Vol. 6, pp 79-98, 1963.

14. L. B. Hofman, "Performance Results for a Hybrid Coding System," Proc. of the International Telemetry Conference, Vol. VII, pp 969-476, Sept. 1971, Washington D.C.

15. L. B. Hofman and J. P. Odenwalder, "Hybrid and Concatenated Coding Applications," to be published in Proc. of the International Telemetering Conference, Vol. VIII, Oct., 1972, Los Angeles, Calif.

16. F. Jelinek, and J. Cocke, "Bootstrap hybrid decoding for symmetrical binary input channels," Information and Control, March 1971.

17. F. Jelinek, "A Study of Sequential Decoding," NASA CR-114450, February, 1972.

18. D. D. Falconer, "A hybrid coding scheme for discrete memoryless channels," Bell System Technical Journal, vol. 48, pp. 691-728, March 1969.